IITK NETWORK -
HARDWARE AND SOFTWARE DEVELOPMENT ON MICRO-78

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

BY

RAKESH KUMAR JAIN

to the
COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JULY, 1980

CSP-1980—M-JAI—HAR

CERTIFICATE

This is to certify that the thesis entitled 'IITK Network - Hardware and Software Development on MICRO-78', which is being submitted by Sri R.K. Jain in partial fulfilment of the requirements for the award of the degree of MASTER OF TECHNOLOGY, has been carried out under my supervision and has not been submitted elsewhere for the award of a degree.

July 1980

A.S. Sethi
Assistant Professor
Computer Science Program
Indian Institute of Technology,
Kanpur

ACKNOWLEDGEMENT

I am extremely grateful to Dr. A.S. Sethi for teaching me computer networks and for his guidance in this project.

I am grateful to Professor V. Rajaraman for helping at various stages during this project work.

I am thankful to Sri Wariyam Singh, Sri V. Gupta, Sri Dureja, Sri S.L. Agarwal and Sri S.A. Siddiqui for their help during the project work.

I am grateful to Sanjiv Kumar in whom I found a very sincere project partner and without whose help and cooperation it would have been very difficult to complete this work.

I take this opportunity to express my gratitude to Sri R. Ramaswamy, Sri A.M. Suthar, Sri Salil Durani, Sri Vinod Gupta and Sri E. Ramesh Narayana, who were always helpful as friends and classmates.

Finally, I thank Sri H.K. Nathani for his excellent typing, Sri Maikoo Lal for the xeroxing and Sri Shankar Bux Singh for the cyclostyling.

Rajain

Kanpur
July 1980

- Rakesh Kumar Jain

ABSTRACT

A project was started at the IIT Kanpur Computer Centre in 1978 to link up DEC-1090, TDC-316 and IBM-1800 in a star configuration using a MICRO-78 as the central switch. In the first phase of the work, hardware interfaces were designed for these machines (except the interfaces for the MICRO to DEC link). In the second phase, i.e., the current one, the hardware interfaces for the MICRO to DEC link have been designed, certain changes have been made in the earlier interfaces and a part of the software has been implemented on MICRO-78 and IBM-1800. This thesis describes the design of the hardware interfaces for the MICRO to DEC link and the software implementation on MICRO-78.

CONTENTS

CHAPTER 1

INTRODUCTION

A computer network is a configuration in which two or more computer systems are interconnected for communication among themselves. Such a configuration has many applications, some of which are resource sharing, remote job entry, and load sharing. Computer networks started coming up in late 60's and today we have got many large networks in the world, a few examples being ARPA, TYMNET, CYCLADES, SITA, ALOHA, PRESTEL and SWIFT. In India, this field is still in its infancy, but of late considerable interest has been shown to develop networks in India and one such attempt has been made at the IIT Kanpur Computer Centre. This thesis describes one phase of the work in developing an experimental computer network at IIT Kanpur.

1-1. IIT/KANPUR COMPUTER NETWORK

The IIT/K network will link up DEC-1090, TDC-316 and IBM-1800 in a star configuration using a MICRO-78 as the central switch. The configuration along with the hardware required to interface them is shown in Figure 1-1. The hardware interfaces 1,2,3,4,5,8,9, 10 and 11 shown in this figure have already been built (Ref. 1 for 1,2,3,4, and 5; Ref. 5 for 8,9; and Ref. 6 for 10,11) and in the present phase of work interfaces 6 and 7 have been designed. These interfaces are on the MICRO-78 side of the MICRO to DEC link. The counterpart of this hardware interface on DEC-system 10 is called DQ 11 (Ref. 3).

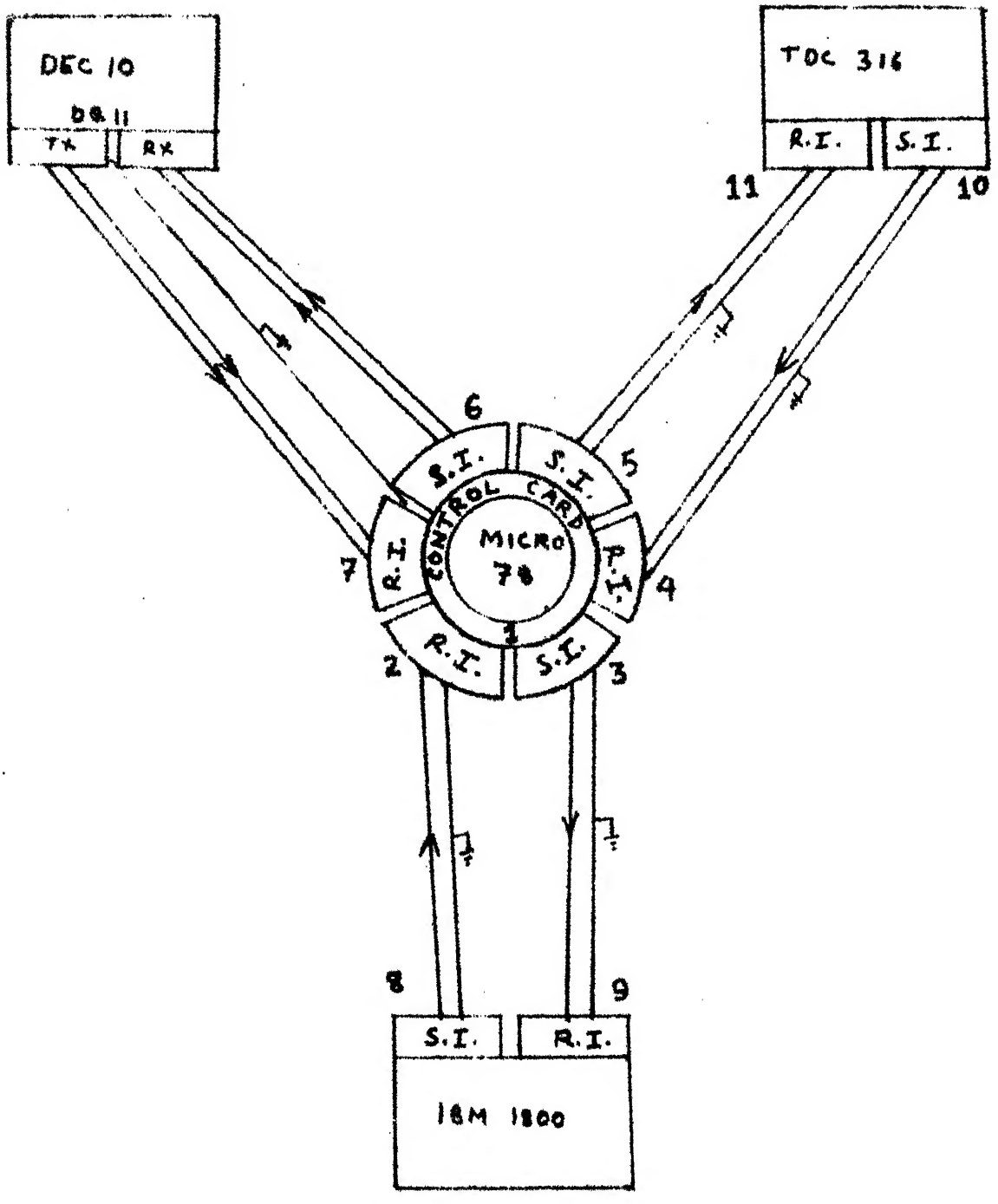

DEC 10
DR 11
TX
RX
TDC 316
R.I.
S.I.
11
10
6
S.I.
S.I.
5
CONTROL CARD
MICRO
78
R.H.
P.T.
7
4
R.I.
S.I.
2
3
8
9
S.I.
R.I.
IBM 1800

FIG. 1.1

The work done in the present phase also includes making some changes in the earlier hardware interfaces and implementing a part of the network software on IBM-1800 and MICRO-78. The network software has been designed according to the DEC specifications since the DEC-system provides comprehensive network facilities. Following is a brief description of the digital network architecture.

1-2. DIGITAL NETWORK ARCHITECTURE

DIGITAL provide networking facilities for their computers and the family of products that create this network is called DECNET. All implementations of DECNET adhere to a common network architecture that defines the structure and protocols used to communicate through the network. This architecture provides a modular design for DECNET and has three distinct layers:

(i) The Physical Link Control Layer
(ii) The Network Service Layer
(iii) The Application Layer.

Each layer has got some well defined functions and is related to layer above it. Following is a brief description of these layers:

(i) The Physical Link Control Layer: This is the lowest layer and lies immediately above the physical transmission of bits over the data link. The functions of this layer are performed by the Digital Data Communications Message Protocol (DDCMP). This protocol is responsible for the correct sequencing and integrity of the data transmitted over a physical link.

(ii) The Network Services Layer: This layer performs the following functions:

(a) Routing the messages between source and destination nodes.

(b) Managing logical data channels, i.e., establishing logical communication links between different processes (e.g., user programs) so that they can exchange information regardless of their physical locations within the network.

The protocl which peforms these functions is called Network Services Protocol (NSP).

(iii) The Application Layer: The functions of this layer are performed by the Data Access Protocol (DAP) which is a user level protocol. Its primary purpose is to permit remote file access within the DECNET environment independently of the I/O structure of the operating system being accessed.

In the current phase of work, only the first layer (DDCMP) has been implemented on IBM-1800 and MICRO-78.

1-3. OVERVIEW OF THE THESIS

Chapter 2 describes in detail the design of hardware interfaces for the MICRO to DEC link. In Chapter 3, DDCMP has been described in detail and in Chapter 4 its implementation on MICRO-78 has been discussed. Conclusions are given in the last chapter.

CHAPTER 2

MICRO-78 HARDWARE INTERFACES FOR THE DEC-10 LINK

In this chapter, we discuss the design of the MICRO-78 Hardware Interface for the DEC-system-10 link. As explained in Chapter 1, the counterpart of this interface on the DEC-10 is the DQ-11 Synchronous Interface on the DN87S Front-end Processor. The design of the MICRO-78 hardware interface for this link has been influenced by

(i) The line protocol used by DQ-11, and

(ii) The line protocol that is used on the MICRO-78 to TDC-316 link or on the MICRO-78 to IBM-1800 link.

The strategy in designing this hardware interface has been to make the minimum possible changes in the already designed interfaces on MICRO-78, for the links to the TDC-316 and IBM-1800 (these two interfaces are identical). The advantage of designing by this strategy is that we can use the same printed circuit boards as are used for MICRO-78 to TDC-316 link (or MICRO-78 to IBM-1800 link) with minor changes on them.

Figure 2-1 shows the basic block diagram of the hardware configuration for the DEC-10 link. As shown in the figure, there is a control card which acts as the interface between the MICRO-78 system and the sending and receiving interfaces. The control card has various functions to perform which are described in Ref. 1, Sec. 4-2.

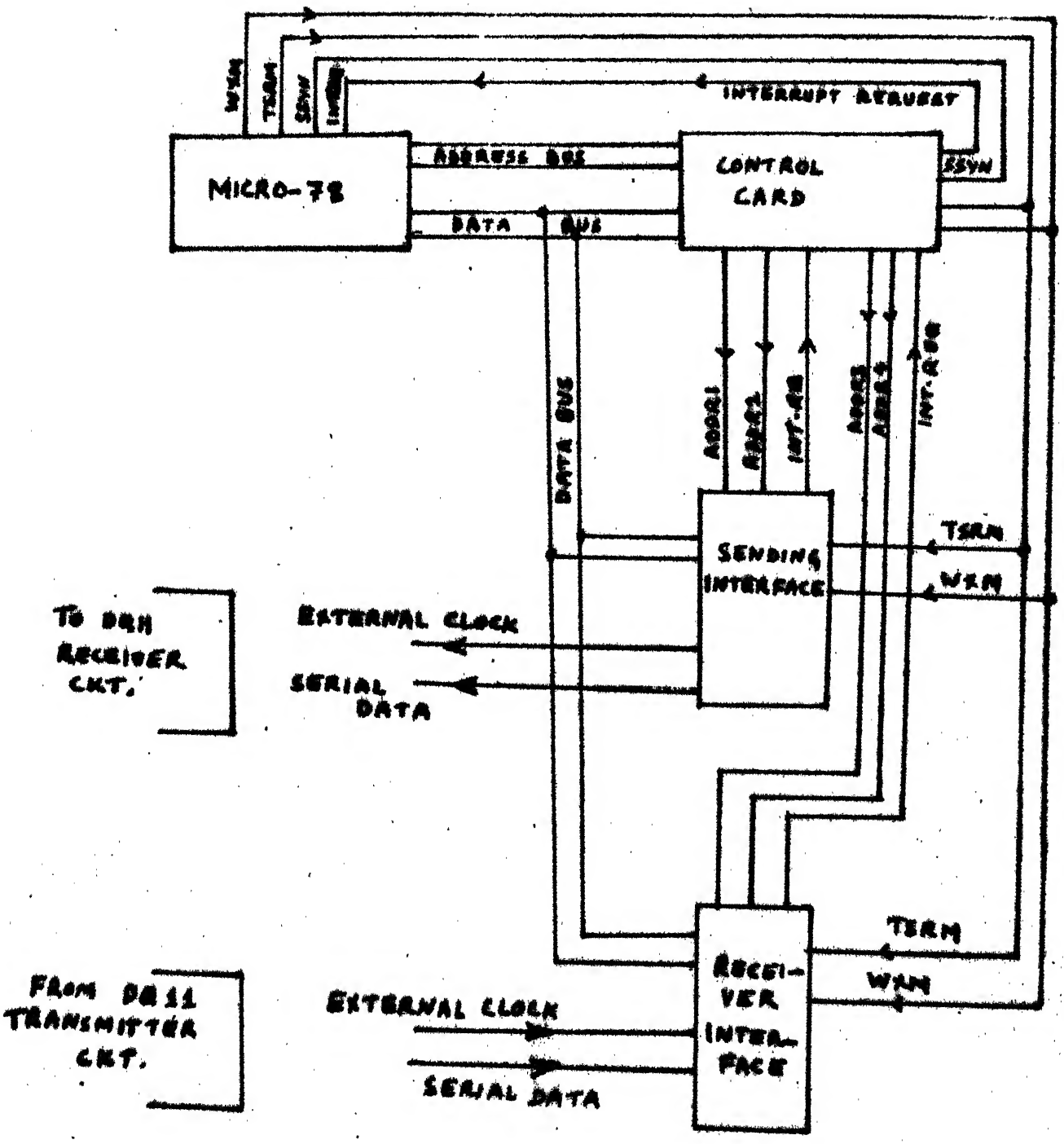

MICRO-78
CONTROL CARD
ADDRESS BUS
DATA BUS
INTERRUPT REQUEST
SYN
DATA BUS
SENDING INTERFACE
TSRM
WSM
TO DR11 RECEIVER CKT.
EXTERNAL CLOCK
SERIAL DATA
FROM DR11 TRANSMITTER CKT.
RECEIVER INTERFACE
TSRM
WSM
EXTERNAL CLOCK
SERIAL DATA

FIG 2.1

2-1. SENDING INTERFACE

The Sending Interface has two addressable registers. One register holds the control and status information and is called control and status register (CSR). The other is the data register (DTRG) which contains the data loaded by CPU to be transmitted serially on the serial data line.

The sending inferface actions can be summarised as follows:

(1) With power on, the CSR bits are cleared, and the sending interface transmits SYN characters (226_8) over the line (this is the idling state).

(2) Processor deposits the first byte of the message to be transmitted in DTRG and sets flags for the initiation of the message (INIT) ~~and byte handed over (TX)~~ in the CSR.

(3) At the end of current byte transmission, the interface checks if at least one SYN has been sent after the last message was transmitted. If so it loads DTRG contents into the parallel to serial converter register (TXRG) whose contents can be shifted out on the line, LSB first.

(4) As soon as data is transferred from DTRG to TXRG an interrupt is generated. The CPU transfers the next byte onto DTRG.

(5) With the interrupt request following the last byte transmission, the CPU resets CSR bits thereby causing SI to go back to Step 1 after the current (last) byte has been transmitted.

2-1.1 Hardware Description of the SI: The CSR is a 5 bit register implemented using D flip flops (2 x 7474). The CSR format is as shown in Figure 2-2. The SI protocol in HDL (hardware description language) is as follows:

Definitions:

Register: DTRG (0-7), CSR (0-7), LC

Shift Register: TXRG (0-7)

Clock PP

Character SYN (226 in octal).

$/(\overline{INIT} + \overline{MODE})$. (LC=0) . PP/TXRG $\leftarrow$ SYN, MODE $\leftarrow$ 1 (S1)

/(LC $\neq$ 0) . PP/Shift right TXRG, decrement LC (S2)

/INIT . MODE (LC = 0) . PP/TXRG $\leftarrow$ DTRG, TX $\leftarrow$ 0 (interrupts processor) (S3)

/INIT . MODE . $\overline{TX}$. (LC = 0) . PP/ERR $\leftarrow$ 1 (S4)

/WXM + DTRG / PRESET TX (S5)

(Note: WXM and DTRG follow negative logic. Thus S5 is equivalent to saying that DTRG has to be written into by the PC).

The expressions enclosed in / / define various combinations of the CSR bits and other conditions. The action following it is the action to be taken when such condition is satisfied. S1, S2, etc., are the control signal numbers which correspond to these conditions.

2-1.2 Explanation of CSR Bits: The CSR bits as depicted in Figur 2-2 have the following significance:

(i) INIT: The initialization bit. It is set by processor to indicate that a message is ready to be transmitted. It is reset when the last byte has been transmitted.

(2) MODE: This is set by interface to indicate that at least one SYN has been transmitted. It is reset at the end of message transmission by PC to enable the transmission of at least one SYN character over the line before the next message starts.

(3) TX: This is the bit which keeps track of bytes as they are handed over to the SI by the processor. The sending interface sets it when a byte is dumped into the DTRG by the processor and resets it when the byte has been transferred to TXRG.

(4) INTF: Interrupt flag. This is set whenever INIT is set and TX is reset, indicating an interrupt condition. Setting of this generates an interrupt and this flag is used to indicate to the processor the nature of the interrupt.

(5) ERR: If with the previous byte transfer TX was reset and the interrupt thus generated was not serviced by the PC in time, then the TX would remain reset at the end of the current LC = 0 cycle This means that the DTRG contents are the same as the old value since new byte has been deposited into it. This is an error condition causing retransmission of a byte and is indicated by this bit. It is reset by PC.

2-1.3 SI Block Diagrams (Figure 2-3): The SI hardware consists of the following functional blocks:

(a) Clock generator
(b) Data register and parallel to serial converter
(c) Control signals
(d) Line driver
(e) CSR

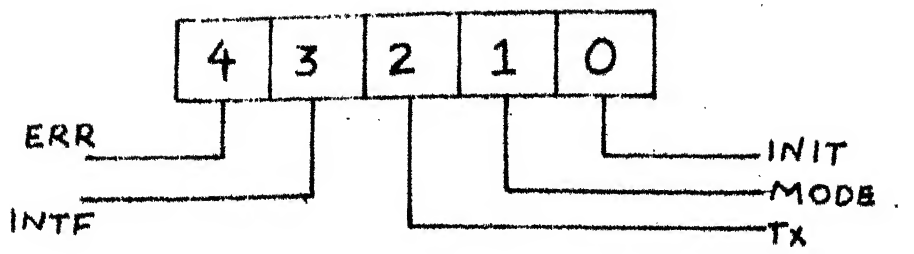


FIG. 2.2 : CSR BITS

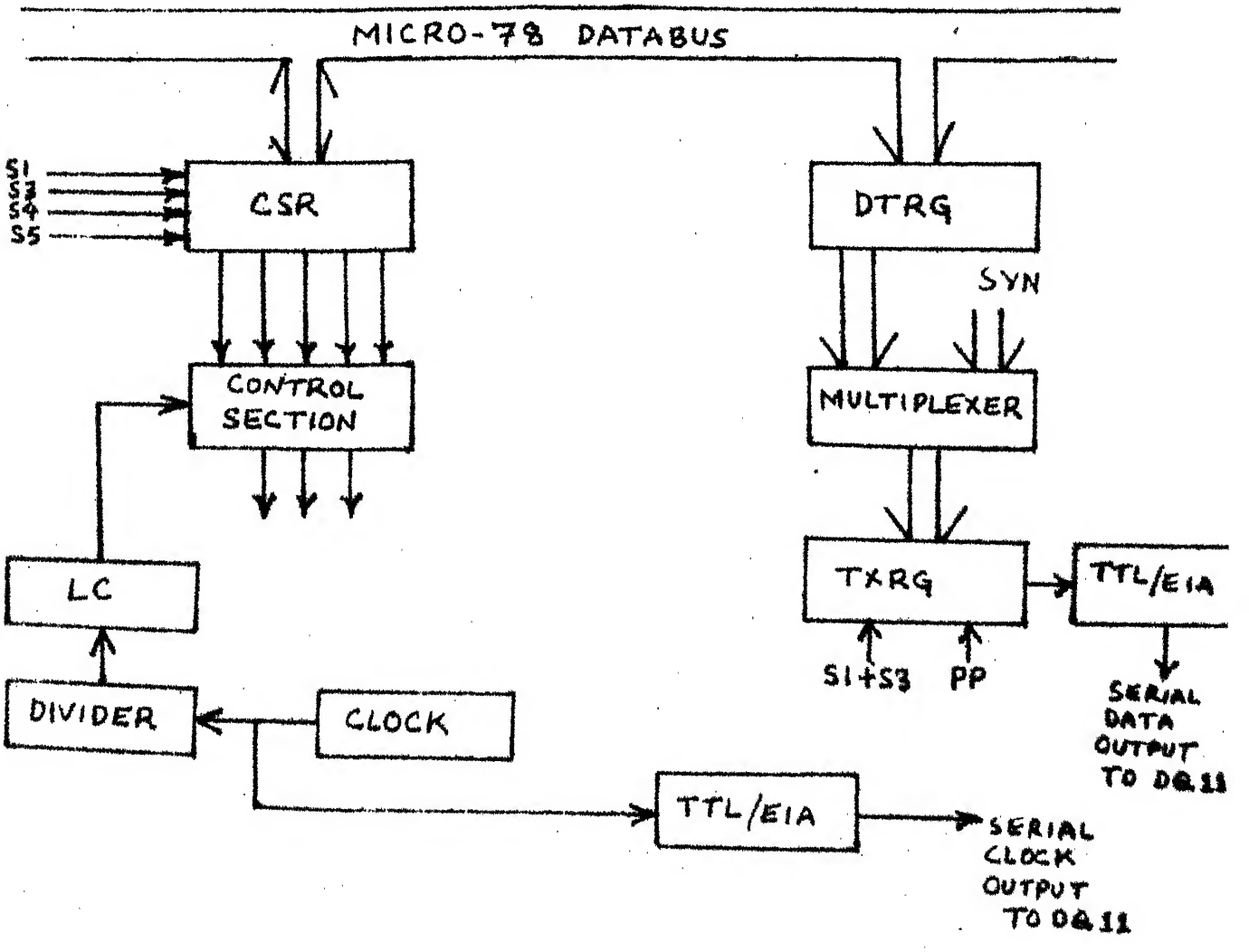


FIG. 2.3 : OVERALL BLOCK DIAGRAM OF TRANSMITTER INTERFACE

The following is a brief description of these blocks.

(a) Clock Generator (Figure 2-4(a)): The clock generator uses a 2 MHz crystal. Motorola's LM375 generates the basic clock which is divided down by a chain of 7490's and a 7474 (Figure 2-4(b)) to get a 10 KHz clock and this clock is called PP.

(b) Data Register and Parallel to Serial Converter (Figure 2-5): The DTRG outputs and the strip switch with SYN combination terminate on the multiplexor, except when S3 is active, SYN combination is selected. Multiplexer output goes to the TXRG input. As per the hardware algorithm, TXRG is to be loaded whenever $S_1 + S_3$ is active. PP does the shifting out. To avoid ambiguity in loading data, $S_1 + S_3$ should be slightly delayed as compared to the switch over leading edge of S_3. This goal is easily accomplished since an extra gate is needed for deriving $S_1 + S_3$ (Figure 2-5(b)).

(c) Control Signals (Figure 2-6): Control signals are all low active since the signals to set/reset 7474's are to be low signals. The generation of control signals is self explanatory.

(d) Line Driver (Figure 2-5(b)): The DQ11 specifications require the data on serial input line to be of EIA specifications. For this purpose TTL/EIA converter is used.

(e) The CSR Controls (Figure 2-7): CPU communication with CSR is similar to that discussed for DTRG. Note in the figure that all the clear inputs except that of TX bit are directly connected to master clear (MC) of the MICRO-78, while the TX bit is cleared by MC + S This facilitates the clearing of all bits when the MICRO-78 is started up (or MC switch on the console of MICRO-78 is pushed). TX bit is

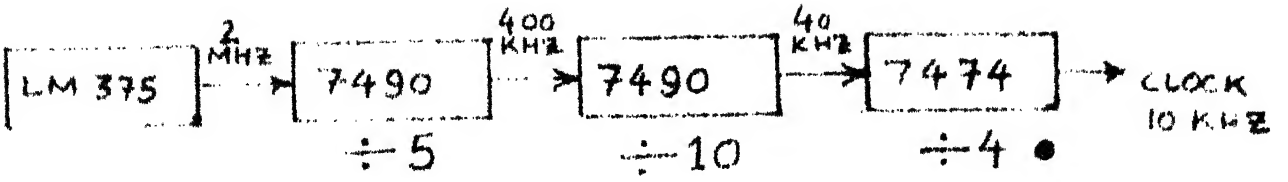


FIG. 2.4(b) DIVIDER CHAIN

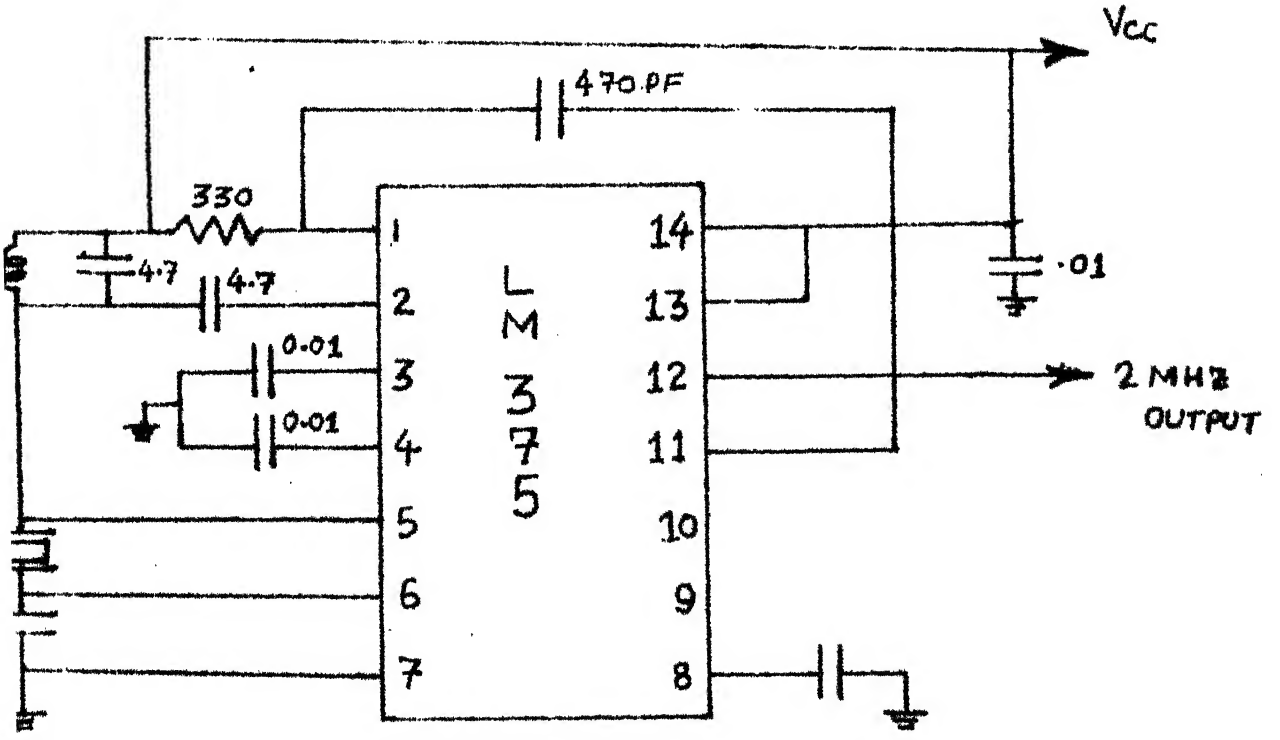
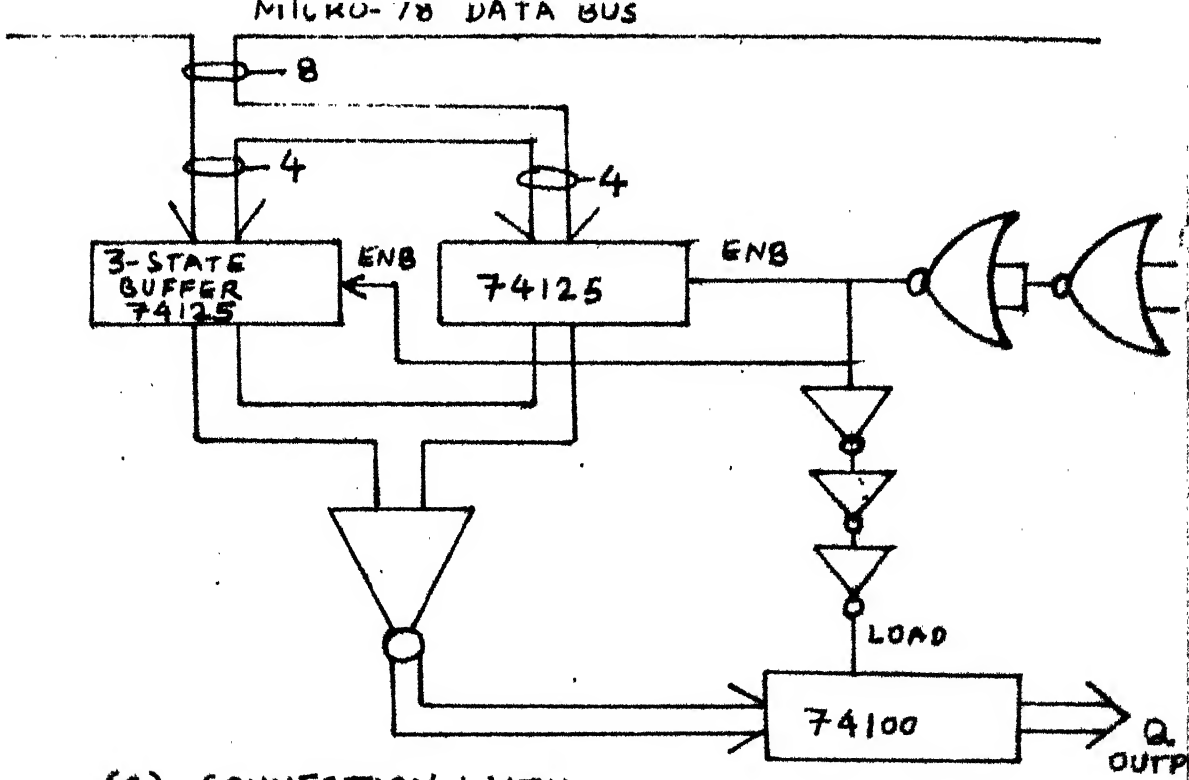
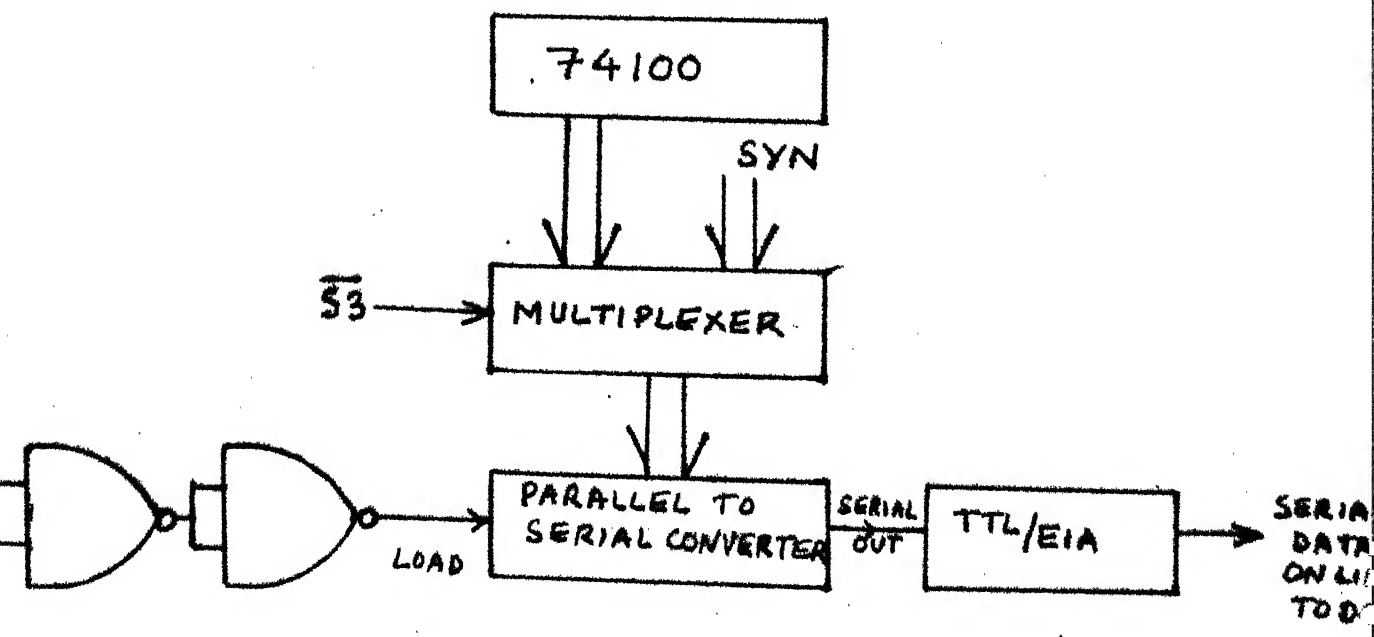


FIG 2.4(a): CLOCK GENERATOR



(a) CONNECTION WITH DATA BUS



(b) DTRG TO TXRG INTERCONNECTION

FIG. 2.5 : DATA REGISTER AND PARALLEL TO SERIAL CONVERTER

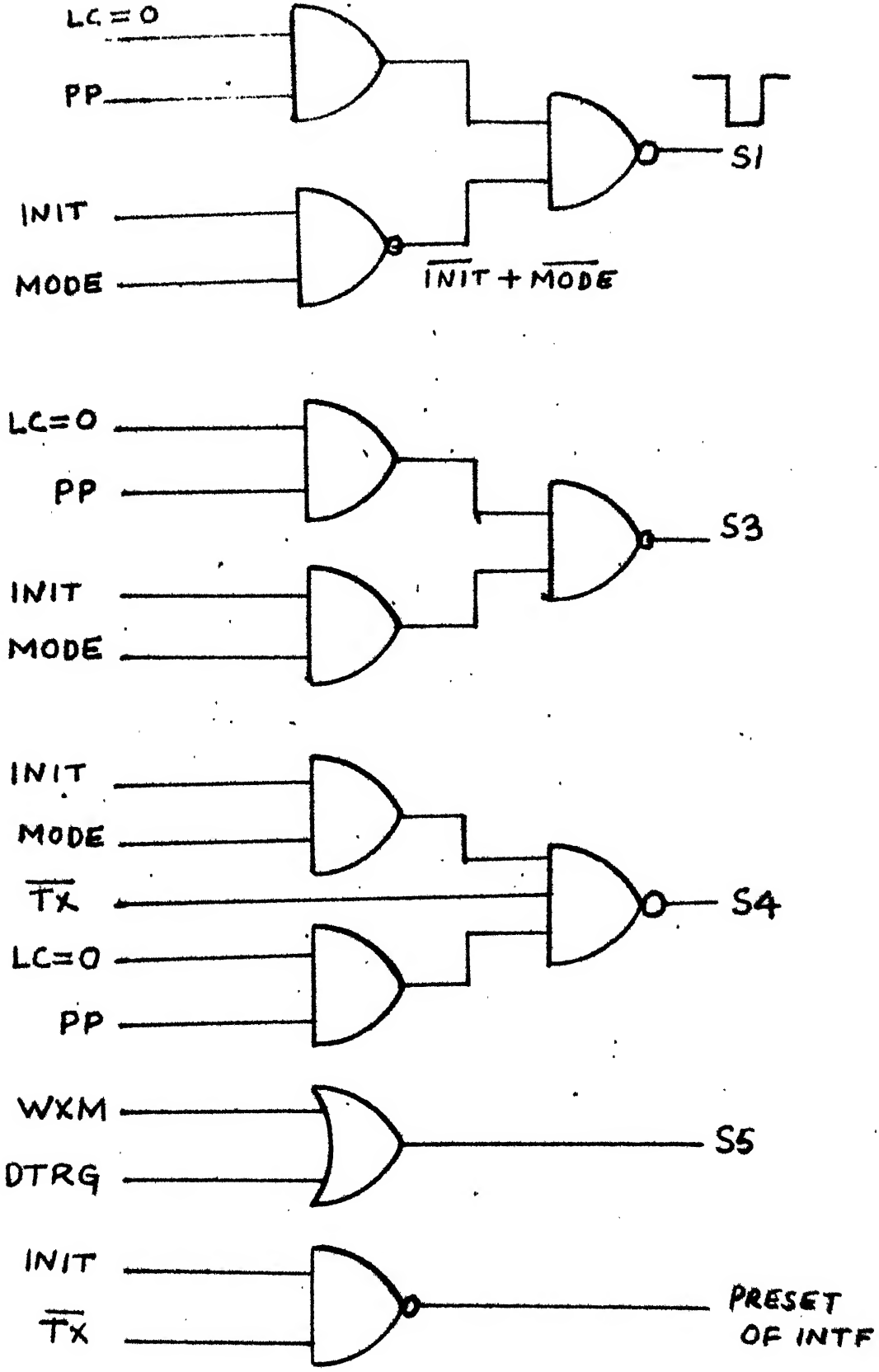


FIG 2.6 : CONTROL SIGNAL GENERATION

cleared by the interface in every byte transfer i.e., when S_3 is present or it can be cleared by the MC.

(f) Line Counter LC (Figure 2-8): Any action by the interface occurs at LC = 0 points. Every time a count is completed, the ripple clock output causes the count of 8 to be loaded in the 74190 and then the count down starts. This serves to mark out the 8-bit periods by giving out Max/Min output which is used as LC = 0 pulse.

(g) Interrupt Generation (Figure 2-9): Whenever the INTF is set an interrupt request is to be transmitted to the control card priority encoder circuit. The priority encoding and servicing of the interrupt is described in Ref. 1 Sec. 4-2.

(h) Transmission of System Clock (Figure 2-3): The system clock is transmitted by the sending interface along with the data to DQ11 external clock input. The clock must conform to EIA specifications and for this purpose TTL/EIA converter is used in the same way as used in transmitting data.

2-2. RECEIVING INTERFACE

The receiving interface has the following functions:

(i) To mark out byte boundaries by detecting a SYN character over the line (byte synchronization).

(ii) To detect ~~SOH~~ non-syn character in the idle and byte synchronized state and generate an interrupt.

(iii) To interrupt PC every time byte is handed over to the DTRG in the message reception mode.

(iv) To set a flag indicating over-run, if any interrupt request is not serviced in time.

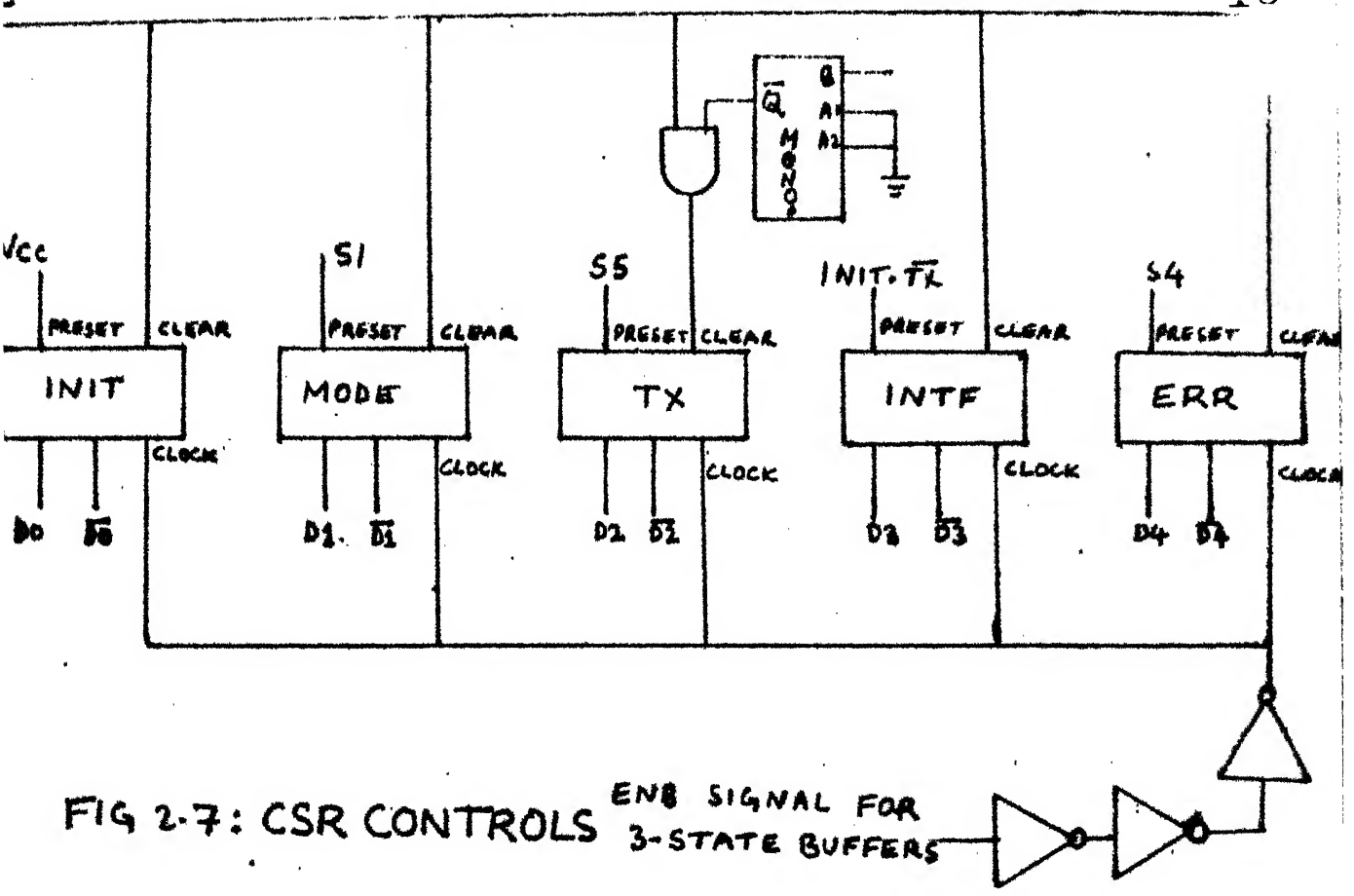


FIG 2.7 : CSR CONTROLS

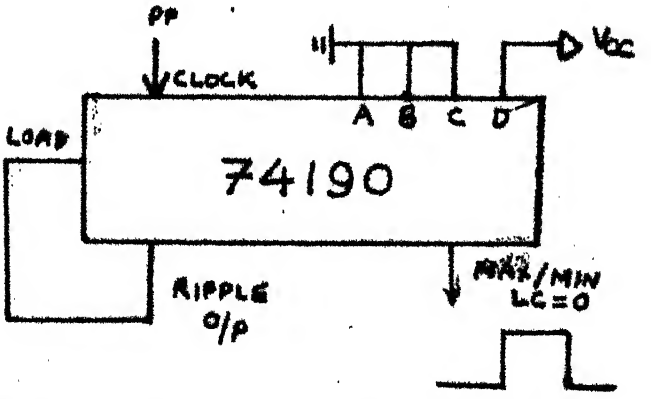
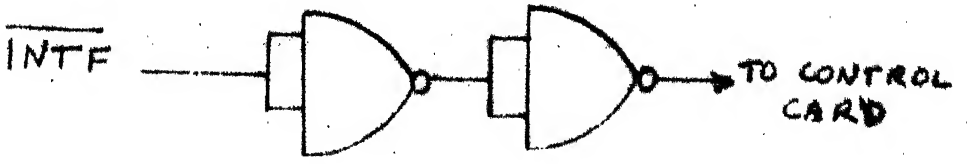


FIG 2.8 : LINE COUNTER



Inputs to the receiving interface are:

(i) Serial data on the line, which conforms to EIA standards. This data is converted to TTL specifications using EIA/TTL converter as shown in Figure 2-10.

(ii) The system clock which is transmitted from the DQ11 and is also EIA specified is converted to TTL specification using EIA/TTL converter as shown in Figure 2-10. This clock will be referred to as PP or the system clock.

2-2-1 Hardware Description of RI: Just as the sending interface, the receiving interface also has got two addressable registers, the DTRG and the CSR. The overall block diagram is shown in Figure 2-10 CSR format is shown in Figure 2-11.

2-2.2 Description of the Receiver CSR:

(1) Mode: This is set by the interface to indicate that a non SYN character has been received in the idle mode. It is reset by the processor.

(2) RX (Byte handover flag): It is set by the interface when a byte has been received and handed over to the DTRG. It is reset by the interface when PC has read the DTRG.

(3) INTF (Interrupt Flag): It is set when Mode and RX flags are set. This causes an interrupt request to be generated. This flag is reset when the interrupt has been serviced by the PC.

(4) OVR (Overrun Flag): This flag is set by the interface to indicate an overfun condition. This is cleared by the PC.

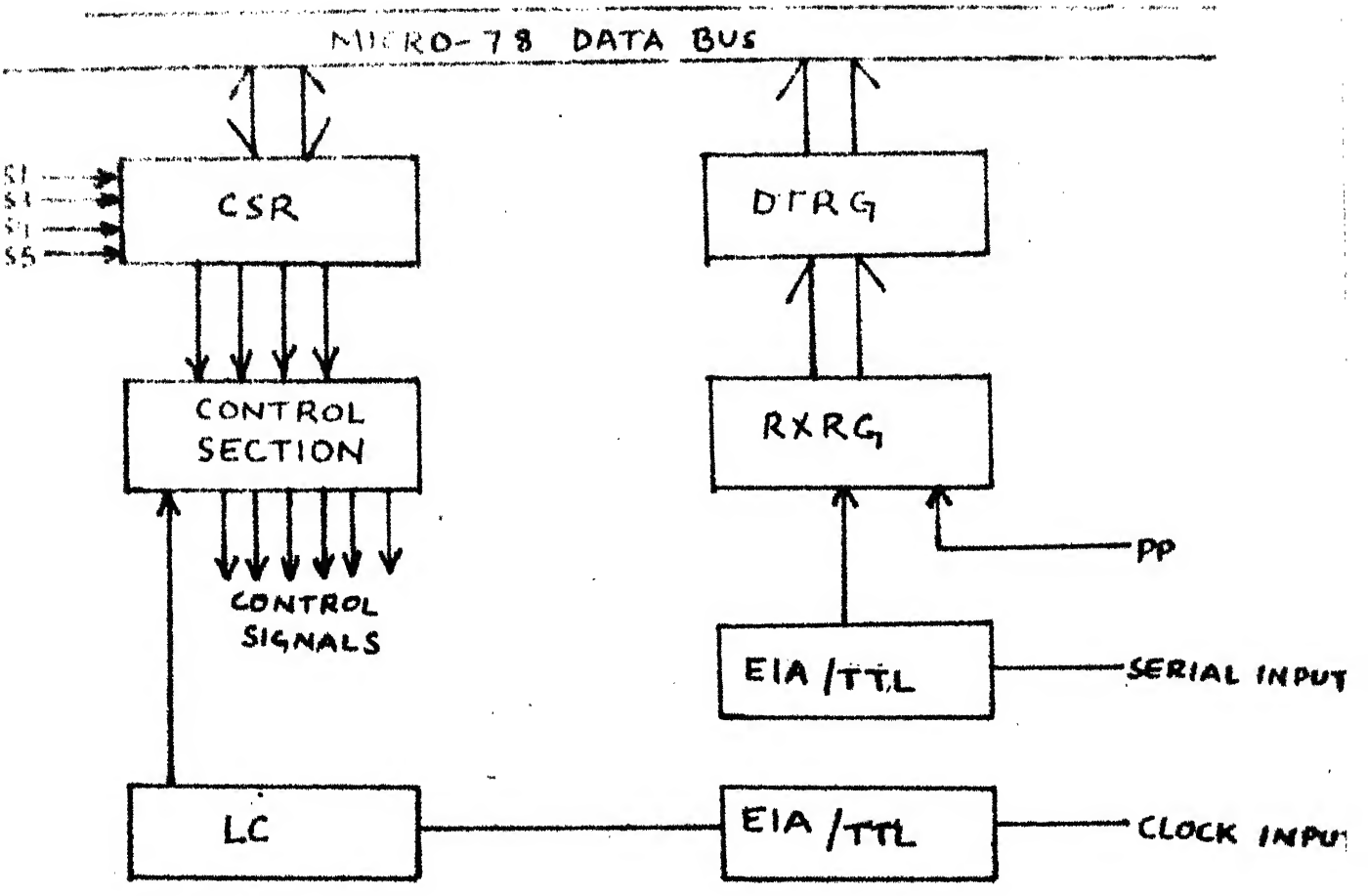


FIG 2.10 : OVERALL BLOCK DIAGAM OF R.I.

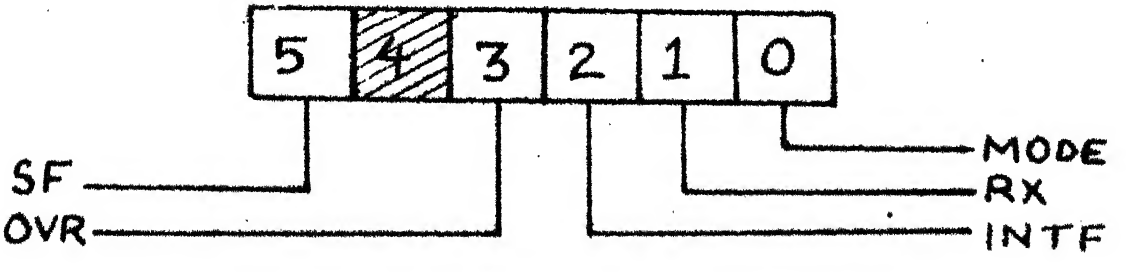


FIG. 2.11 : CSR FORMAT

(5) SF (Byte Synchronization Flag): It is set by the interface when a SYN is recognized for the first time after the mode bit has been reset. It is reset by the processor under various conditions such as:

(i) The first byte of the message is not one of SOH (226_8), ENQ (005_8) or DLE (220_8).

(ii) A message has been completely received, or

(iii) Any other reason governed by the software protocol.

The hardware algorithm for the receiving interface is as follows:

/$\overline{SF}$. PP/ if R X RG = SYN then SF$\leftarrow$1, LC = 8 (S1)

/PP/ Shift right RXRG, RXRG(8) bit from line; decrement LC (S2)

/SF . $\overline{MODE}$. (LC=0) . PP / if RXRG $\neq$ SYN then MODE$\leftarrow$1, DTRG$\leftarrow$RXRG, RX$\leftarrow$1 (S3)

/SF . MODE . (LC = 0) . $\overline{RX}$. PP/DTRG$\leftarrow$RXRG, RX$\leftarrow$1 (S4)

/SF . MODE . (LC = 0) . RX . PP/ OVR$\leftarrow$1 (S5)

/TSRM + DTRG/ Clear RX (S6)

(Note: TSRM and DTRG follow negative logic and S6 is equivalent to saying that the DTRG has been read by the PC).

RXRG is the serial to parallel converter shift register that receives the serial data from the EIA/TTL converter, LC is the line counter. When SF is set, LC is initialized to 8 and for every subsequent LC = 0 tob it is loaded with 8. Thus once set, subsequent LC = 0 would mark out the byte boundaries.

2-2.3 RX Interface Block Diagrams: The hardware consists of following functional blocks:

(a) Data register and serial/parallel converter.

(b) Data SYN detector.

(c) Control Signal Generator

(d) CSR control.

The following is a brief explanation of these blocks:

(a) Data Register (DTRG) and the Serial/Parallel Converter (RXRG)(Figure 2-12): Serial data from EIA/TTL converter gets clocked into the RXRG so that at the end of 8 system clocks one byte of data is in the RXRG. The data is transferred into the DTRG whenever S_3 or S_4 is active.

(b) SYN Detector (Figure 2-13): Three 3-line to 8-line demultiplexers are used for this purpose. The RXRG outputs are connected to the inputs of the demultiplexers as shown. A is the LSB. The output of the detector is low whenever a SYN combination is present at the RXRG.

(c) Control Signal Generator (Figure 2-14): The control signal generation with the help of hardware description and Figure 2-14 is self-explanatory.

(d) CSR Control (Figure 2-15): The CSR design philosophy is the same as in the sending interface and the CSR is accessible to the PC via the 3-state buffers.

(e) Interrupt Generation (Figure 2-16): The INTF flag is set by the interface to interrupt the PC. The INTF flag goes to the control card and an interrupt request is sent to the PC via the priority encoder circuit (discussed in Ref. 1, Section 4-2). The INTF flag is reset by the processor.

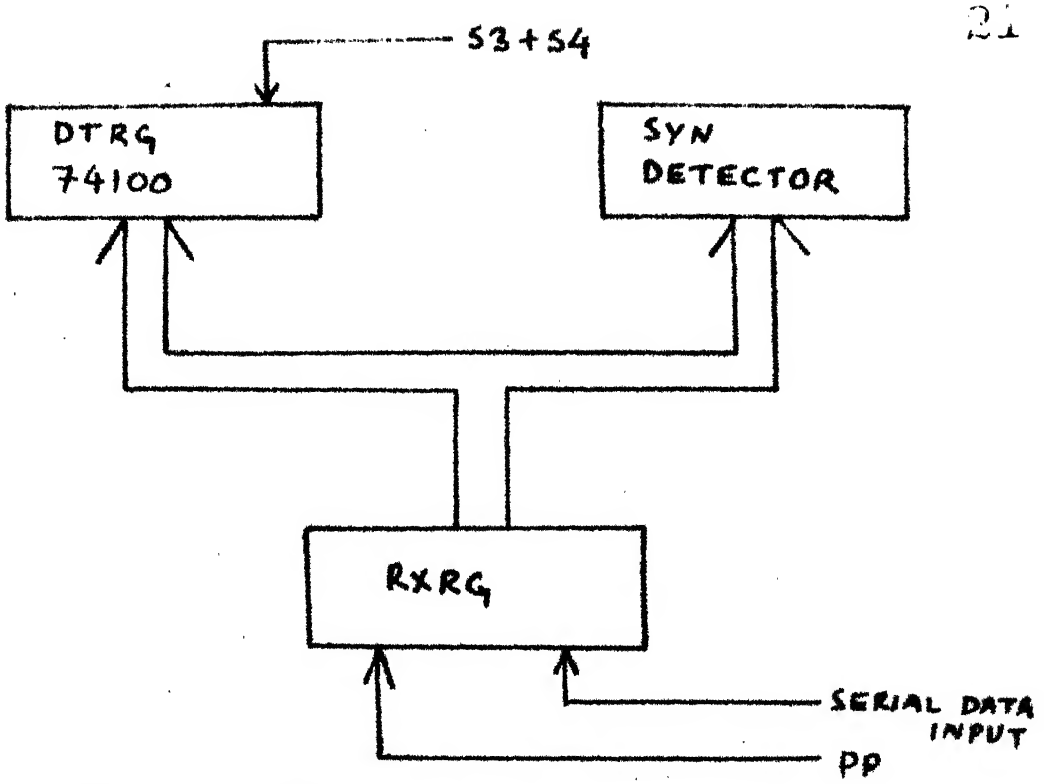
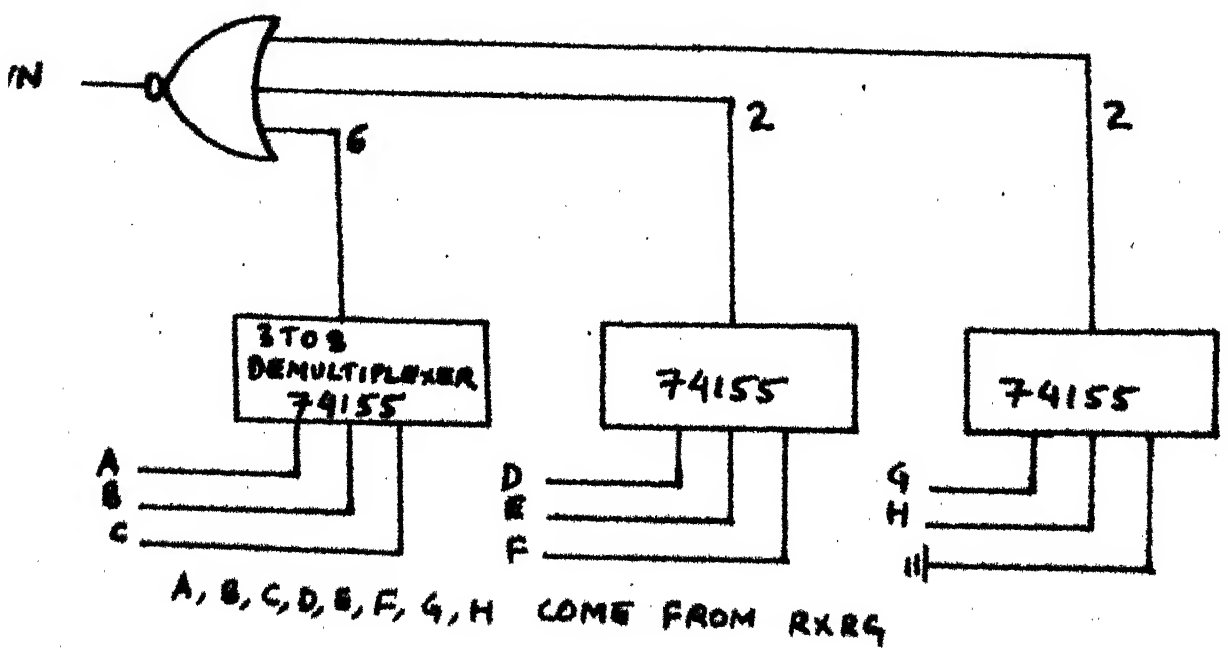


FIG. 2.12 : DTRG AND RXRG CONNECTIONS



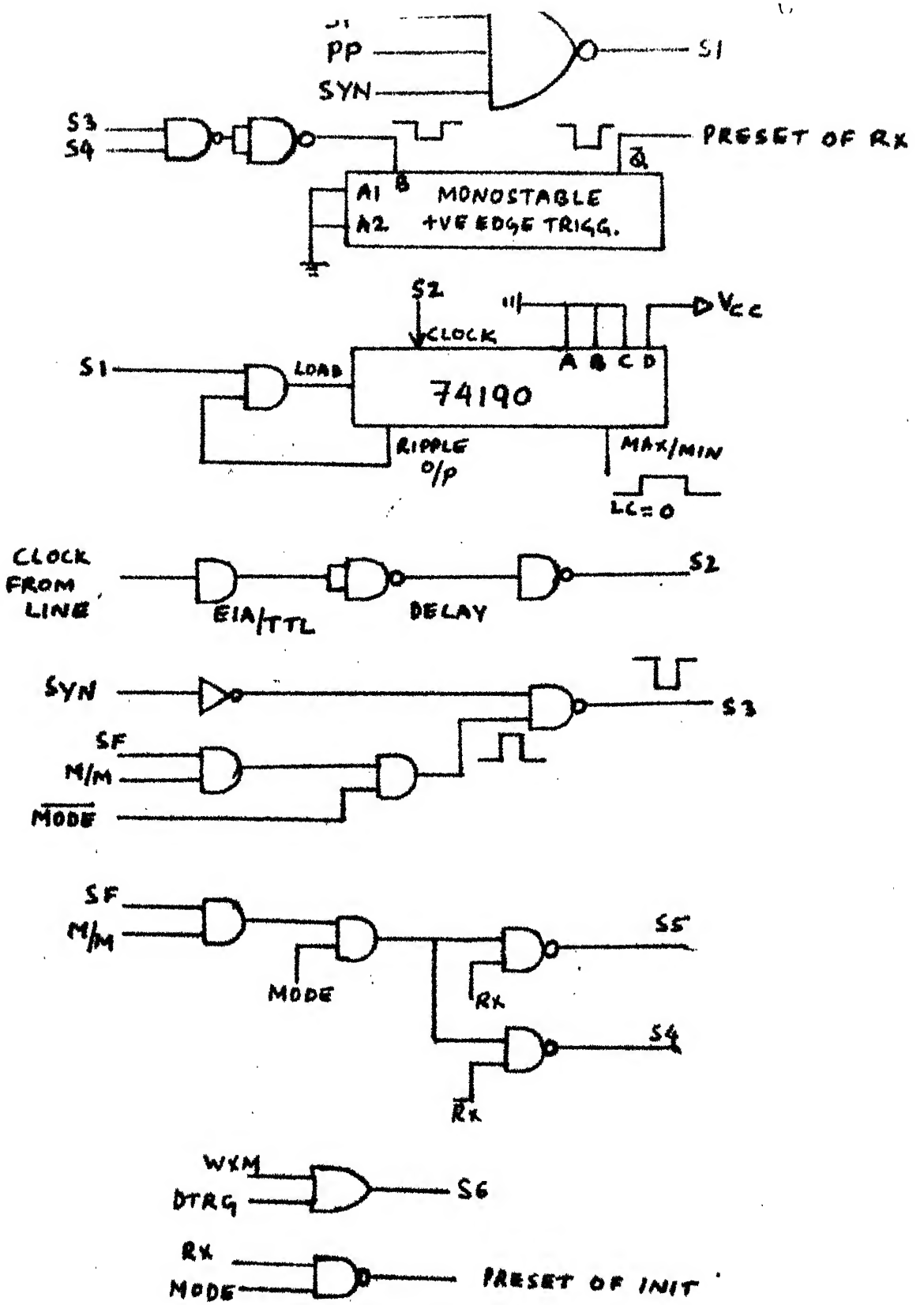


FIG. 2.14 : CONTROL CIRCUITS

CHAPTER 3

DESCRIPTION OF DDCMP

In Chapter 1 (Sec. 1-2), the Digital Network Architecture (DNA) was discussed and it was mentioned that DDCMP forms the lowest layer, i.e., the physical link control layer of the DNA. The main function of DDCMP is to ensure correct sequencing and integrity of the data transmitted over a physical link. This chapter discusses DDCMP in detail.

Please note that -

(i) The term 'message' used throughout is the same as the term 'packet' used in common literature.

(ii) Although DDCMP caters for both synchronous and asynchronous lines, we shall describe here only the portion suitable for synchronous links.

3-1. MESSAGE FORMATS

In DDCMP, we have three kinds of messages -

(i) Data Messages
(ii) Control Messages
(iii) Maintenance Messages

Every data message carries a number (from 0 to 255) with it to enable correct sequencing at the receiving end. The numbering begins with number 1 after initialization (the details of initialization are described in Section 3-3) and is incremented by one (modulo 256) for each subsequent data message. The receiving station acknowledges the correct receipt of data messages by sending an ACK (a control message) or by sending this information along with a data message going towards that station (a piggybacked ACK). The

acknowledgement carries a number 'n' which indicates correct reception of all messages upto & including data message number 'n'. If an acknowledgement is not received within a certain time, a timeout is said to have occurred and a control message REP is sent to enquire about the status of the sent data message. Reply to REP is either an ACK or a NAK (negative acknowledgement). A NAK (another control message) is sent when a message in error is detected. STRT and STACK are two other control messages and are used for initialization of the protocol. Maintenance messages are used for diagnostic testing, bootstrapping, dumping and other such functions.

Now we proceed to describe the message formats in detail.

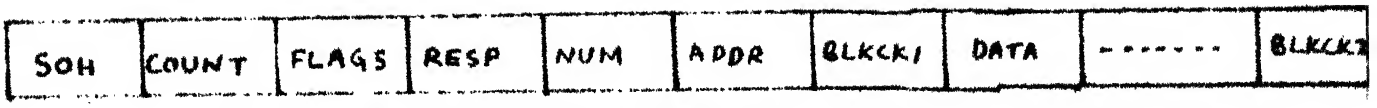
3-1.1 Data Messages: The format of a data message is shown in Figure 3-1(a)

(1) SOH: (1 byte) (Start of Header): This field contains the number 129(201_8) and it identifies a data message.

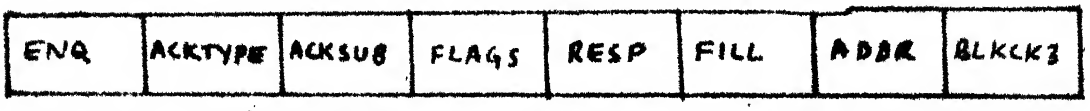
(2) COUNT (14 bits): This field specifies the number of bytes in the data field. The value '0' is not allowed.

(3) Flags (2 bits)

(i) Bit 0: The quick sync flag (Q SYNC flag): When this flag is on, it indicates to the receiver that the next message will not be contiguous to this one and resynchronization should follow this message. The function of this flag will be described in detail in Section 3-2.1.



a) DATA MESSAGE FORMAT

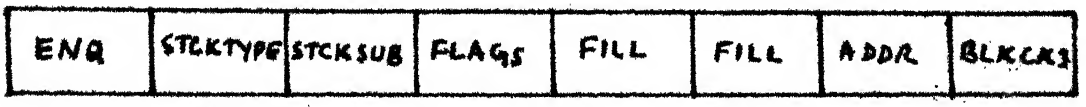


b) ACK MESSAGE FORMAT

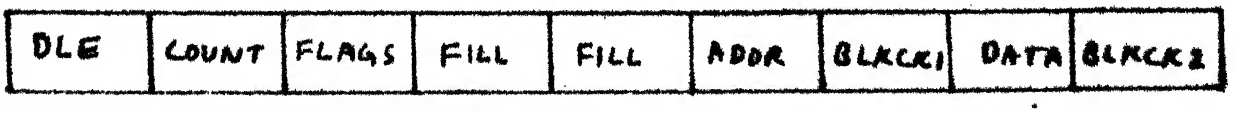
c) NAK MESSAGE FORMAT

d) REP MESSAGE FORMAT

e) STRT MESSAGE FORMAT



f) STACK MESSAGE FORMAT



g) MAINTENANCE MESSAGE FORMAT

FIG. 3.1 : MESSAGE FORMATS

(ii) Bit 1: The SELECT flag. This is used to control transmission ownership on multipoint and half duplex links. In our implementation since we have only full duplex point to point lines, we will not be concerned with this flag.

Note: COUNT and FLAGS form a 2-byte quantity. The first byte contains the 8 low-order bits of COUNT. The second byte contains the SELECT flag in the most significant bit, the QSYNC flag in the next bit and the 6 higher-order bits of COUNT in the remaining 6 bits.

(4) RESP (1 byte): This is the piggybacked ACK field and carries the number of the last consecutive correct message received from the other end by the station transmitting this message. It implies that all unacknowledged messages between the one acknowledged in the last RESP field and the one acknowledged by this RESP field (modulo 256) have been received correctly.

(5) NUM (1 byte): This field carries the number of this data message.

(6) ADDR (1 byte): This is of relevance only on multipoint links and carries the address of the tributary stations on multipoint links. This field is of no concern to our implementation.

(7) BLKCK1 (2 bytes): These 2 bytes carry the block check on the header (SOM through ADDR) using the CRC-16 polynomial $x^{16}+x^{15}+x^{2}+1$. The block check is transmitted x^{15} bit first. On reception the CRC computation should yield a zero remainder if no errors exist.

(8) Data("COUNT" No. of Bytes): This is the data field and carries as many bytes as specified in the COUNT field. This field is totally transparent to the protocol and has no restrictions on

bit patterns, groupings or interpretations.

(9) BLKCK2 (2 bytes): This field contains the block check on the data using the same polynomial as for BLKCK1.

3-1.2 Control Messages: There are 5 types of control messages: ACK, NAK, REP, STRT and STACK. We shall describe them one by one:

(1) ACK: As already mentioned, ACK message acknowledges the correct receipt of data messages. It conveys the same information as the RESP field in a data message and is used when acknowledgements are required but there is no data message to be sent in the reverse direction. Its format is shown in Figure 3-1(b).

(1) ENQ (1 byte): This field is the identifier of a control message and has value 005.

(2) ACKTYPE (1 byte): This identifies the type ACK and has value 1.

(3) ACKSUB (6 bits): This field has a value of '0'.

(4) FLAGS (2 bits): These flags are the QSYNC and SELECT flags and have the same function as in data messages.

(5) RESP (1 byte): This field has the same function as the RESP field of data messages.

(6) FILL (1 byte): This byte has value '0'.

(7) ADDR (1 byte): Same as the ADDR field of data messages.

(8) BLKCK3 (2 bytes): Block check on the control message (on ENQ through ADDR). Similar to BLKCK1 for a data message.

(ii) NAK (Negative Acknowledgement): This message is sent to the other end when an erroneous message is received from that side. The reason for error is also sent. It also includes the same

information as carried by an ACK message and thereby serves two functions: acknowledging previously received correct messages and notifying some error condition. The format for this message is shown in Figure 3-1(c). The ENQ, FLAGS, FILL, ADDR and BLKCK3 fields are the same as in an ACK message.

(1) The field NAKTYPE (1 byte): indicates the type NAK and has a value of 2.

(2) The field REASON (6 bits) identifies the source and reason for error.

The various values and the corresponding reasons are listed below:

Value	Reasons
1	Header block check error
2	Data field block check error
3	REP response
8	Buffer Temporarily unavailable
9	Receiver overrun
16	Message too long
17	Message header format error.

(3) The RESP field is the same as the RESP field of ACK and data messages, and in NAK it usually implies some error in the message with number RESP+1 (mod. 256) or beyond.

(iii) REP (Reply to Message Number): This message is sent to enquire about the status of a previously sent message and is usually sent when an ACK or NAK is not received within a timeout period. The reply to a REP is an ACK or a NAK depending on whether the receiving station has received all messages previously sent. The format of a REP is shown in Figure 3-1(d). The ENQ, FLAGS, ADDR and BLKCK3 are the same as in other control messages.

(1) REPTYPE (1 byte): This field indicates the type REP and has a value of 3.

(2) REPSUB (6 bits): This field has a value '0'.

(3) NUM (1 byte): This is the number of the last sequential numbered data message (not including retransmissions) sent by the transmitting station. This is compared against the number of the last sequential message received by the receiving station and results in an ACK if they agree and in a NAK otherwise.

(iv) STRT (Start Message): This message along with STACK is used for protocol initialization and its function will be described in detail under 'start up procedure' in Section 3-3.

Its format is shown in Figure 3-1(e). The ENQ, ADDR and BLKCK3 fields are the same as in other control messages.

(1) STRTTYPE (1 byte): This indicates the type STRT and has a value 6.

(2) STRTSUB (6 bits): This field has a value '0'.

(3) Flags (2 bits): These flags have the same function as in other messages but both flags are ones for STRT message.

(4) FILL: These 2 FILL bytes have a value of '0'.

(v) STACK (Start Acknowledgement Message): This message is sent as a response to STRT during initialization. Its format is shown in Figure 3-1(f). ENQ, FLAGS, FILL, FILL, ADDR and BLKCK3 fields are same as in STRT.

(1) STCKTYPE (1 byte): Indicates STACK type and has value '7'.

(2) STCKSUB (6 bits): Has a value of '0'.

3-1.3 Maintenance Messages: The DDCMP protocol operates in two basic modes: (1) On line or normal running mode, (2) Off line or the maintenance mode. The previous messages correspond to the on line mode. The maintenance mode may be used for basic diagnostic testing and simple operating procedures such as bootstrapping, down line loading and dumping. It provides a basic envelope compatible with DDCMP framing, link management and CRC check for bit errors but does not include any error recovery, time outs or sequence checks.

The format of a maintenance message is shown in Figure 3-1(g).

(1) DLE (1 byte): This field identifies a maintenance message and has the value $144(220_8)$.

(2) COUNT (14 bits): This field specifies the number of 8 bit bytes in the DATA field. The value '0' is not allowed.

(3) FLAGS (2 bits): Both these flags have value '1'.

(4) FILL: These 2 fill bytes have a value of '0'.

(5) ADDR (1 byte): Same as for other messages.

(6) BLKCK1 (2 bytes): CRC check bytes for the header.

(7) DATA ('COUNT' No. of bytes): This is the data field and contains number of data bytes specified in the COUNT field.

(8) BLKCK2 (2 bytes): CRC check on the data

3-2. OPERATION OF DDCMP

The functions of DDCMP can be grouped under three heads:

(1) Framing
(2) Link Management
(3) Message Exchange

These are discussed in the following subsections.

3-2.1 FRAMING: Framing is the process of locating the beginning and end of a message at the receiving end. For this, the receiving end must synchronize at the bit, byte and message levels. We shall describe how DDCMP accomplishes these functions.

(1) Bit Synchronization: This function is achieved by the hardware interfaces (or modems) and is not a part of DDCMP.

(2) Byte Synchronization: Data is sent over the line as a sequence of bits grouped into 8-bit bytes. The receiver must be able to locate the proper 8 bit window in the bit stream and stay in step with it for every subsequent 8 bit grouping. On a synchronous line, this function is accomplished by searching for a unique byte called a SYN byte (value 226_8). Once this is found every subsequent group of 8 bits forms a byte.

DDCMP specifies that the receiver must locate and lock on to 2 consecutive SYN bytes to achieve byte synchronization. The transmitter must send four or more SYN bytes to allow for the loss from missynchronization and hardware interface constraints.

After the reception of a message, two possibilities exist:

(i) The next message immediately follows the current message. In this case the receiver will remain synchronized and reception next of the message will continue.

(ii) The next message does not immediately follow this message. In this case, byte synchronization is assumed to be lost and the receiver must resynchronize for receiving the next message. Hence the next message must be preceded by a synchronization sequence (i.e., 4 or more SYN bytes).

On some communication interfaces like the DMA type, the received data may be buffered in the device and by the time the software driver comes to know that the next message is not contiguous, the device may have buffered many bytes further ahead on the link. So, a long synchronization sequence is required in such cases to account for potential buffering in the interface. The length of the SYN sequence should be 4 more than the number of bytes buffered. A value of 8 is generally suitable.

To reduce the length of the synchronization sequence, the QSYNC flag is used (This is one of the link flags described in Section 3-1.1). When this flag is '1', it notifies the receiver that the next message will not be contiguous to the current one and the synchronization sequence preceding that message may be the short sequence. So, on seeing a QSYNC flag the software driver at the receiver can start resynchronizing immediately after the current message without looking ahead into the next message, thus enabling the receiver to synchronize on a short sequence.

(3) Message Synchronization: The beginning of a message is located by searching for one of the three bytes SOH, ENQ and DLE (which are the starting bytes of data, control and maintenance messages respectively, as described in Section 3-1). One of these bytes must appear immediately after the SYN sequence or immediately after the previous message's last byte. Otherwise, the byte synchronization is assumed to be lost. After detecting the first byte, the end of the message is determined by the following rules:

(1) If the starting byte is SOH or DLE then the next 5 bytes will complete the message header, followed by 2 bytes of CRC check on header followed by 'COUNT' number of bytes of data followed by 2 bytes of data block check. (See Section 3-1).

(2) If the starting byte is ENQ then the next 5 bytes will complete the message followed by 2 bytes of CRC check.

Since no pattern searching is done once the starting byte is found, the data field is totally transparent.

The receiver tries to achieve message synchronization only under the following conditions:

(1) Initially on start up
(2) If messages are not contiguous
(3) If the QSYNC flag is set in the current message, resynchronization is done at the end of this message.
(4) If CRC error or other error occurs that might have caused synchronization to be lost.

In a simple implementation, synchronization may be done after every message, but this is somewhat less efficient.

3-2.2 Link Management: The SELECT flag in all the messages is meant for link management, i.e., controlling the transmission ownership on half duplex and multipoint lines. However, in the case of full duplex point to point lines, there is no link management required. In this case the address field is kept '1' and the SELECT flag has no role to play. We shall not discuss this component of DDCMP since in our implementation only point to point full duplex links are involved.

3-2.3 Message Exchange: This component of DDCMP caters for the exchange of error-free and correctly sequenced data messages. The following is a description of this component of DDCMP.

(1) The transmitter assigns the next sequential message number n to the data message, adds a CRC block check and sends it. After sending a message, a timer is started.

(2) The receiver, after receiving the message, checks it for any errors and compares the message number with the next expected and takes the following actions:

(i) If the number is correct a positive acknowledgement (ACK) carrying this number is sent back and the next expected number is incremented modulo 256.

(ii) If the message is in error, a negative acknowledgement with error reason is sent. The NAK also carries with it the number of the last correctly sequenced message received. NAK is sent to hasten the process of re-transmission, without waiting for a time out. A NAK not only indicates an error, but also acknowledges the correct reception of messages upto the number carried by it.

(iii) If the message is correct but its number is not equal to the expected message number, it is simply ignored.

(3) The transmitter follows the following procedure:

(i) If it receives a positive acknowledgement, it compares its number with the one expected. If it agrees, the message is released, the user is notified and the timer is stopped. If the acknowledgement number does not agree with the expected number, it is ignored.

(ii) If it receives nothing and a timer expires, it sends an enquiry message called REP to the receiver with the number of the message previously sent. The response to a REP is an ACK or a NAK depending on whether the message with that number was correctly received or not. The transmitter timer is again started after sending a REP and if it expires again, this process is repeated.

After some specified number of timeouts, the user is informed, who may decide that the link is out of service.

The following points may be noted about this procedure:

(1) The transmitter can send upto 255 messages without waiting for their ACKs. An ACK confirms the correct reception of all messages between the last acknowledged message and the one with the number contained in this ACK. Same is the case for a NAK. If an ACK gets corrupted, the information lost in it is automatically included in a subsequent ACK, thus eliminating the sending of REP messages if the ACKs are received prior to the expiration of the transmitter timer.

(2) An ACK need not be sent separately, but can be piggybacked with a message going in that direction (the RESP field contains the ACK number).

3-2.4 Message Exchange Variables: The following variables are used in the message exchange state tables. Arithmetic operations and comparisons for these variables are always modulo 256.

(1) N: It is the number of the highest sequential data message transmitted by this station. It is sent in the NUM field of REP messages.

(2) R: The number of the highest sequential data message received at this station. Sent in the RESP field of data messages, ACKs and NAKs.

(3) A: The number of the highest sequential data message that has been acknowledged to this station.

(4) T: It is the number of the next data message to be transmitted. When sending a new data message, T will have the value N+1. When retransmitting, T will be set to A+1 and will advance to N+1.

(5) X: The number of the last data message that has completed transmission.

3-2.5 Control Flag Variables: There are three control flags which specify the sending of control messages.

(1) SACK (Send ACK Flag): This flag is set when either R is incremented (i.e., a new data message has been received) or a REP message is received which requires an ACK reply. It is cleared when a DATA message is sent with a piggybacked ACK, an ACK message is sent or the SNACK flag is set.

(2) SNAK (Send NAK flag): This flag is set when an erroneous message requiring a NAK reply is received. It is cleared when a NAK is sent or when SACK is set.

(3) REP (Send REP flag): This flag is set when a reply timer expires in the running state indicating that a REP should be sent.

3-3. DDCMP STATES AND THE START UP PROCEDURE

DDCMP can be in one of the following states:

(1) HALTED: This state signifies that the protocol is not running.

(2) STARTING: This means that an attempt is being made to initialize the protocol. This state has two internal states, ISTRT and ASTRT, the details of which will be seon made clear.

(3) RUNNING: This is the on line state of DDCMP in which numbered data messages are exchanged.

(4) MAINTENANCE: This is the off line state of DDCMP in which maintenance messages are exchanged.

From the halted state, protocol initialization begins when a start up command is given by the user. An exchange of STRT-STACK messages takes place and state transitions take place from HALTED to RUNNING via ISTRT and ASTRT. These transitions are depicted in the state diagrams shown in Figure 3-2.

3-4. THE RUNNING STATE AND DATA TRANSFER

It is the RUNNING state in which numbered data messages are exchanged, acknowledgements sent, retransmissions done and all similar actions taken. Following is a table showing various events and corresponding actions in the RUNNING state. Note that all comparisions are modulo 256. The priority of transmission is given as: NAK, REP, data message, ACK.

Event	Action
•Receive data message (NUM=R+1)	Give message to user, set SACK flag.
•Receive data message (NUM≠R+1)	Ignore
•Receive message with errors	SET SNAK flag with appropriate reason
•Receive REP (NUM=R)	SET SACK
•Receive REP (NUM≠R)	SET SNAK, reason 3.
•Receive ACK or data message with $A < RESP \leq N$	For all messages $(A < NUM \leq RESP)$, notify user of their receipt and release them. $A \leftarrow RESP$ If $T \leq A$, then $T \leftarrow A+1$ If $A < X$, start timer If $A \geq X$, stop timer
•Receive ACK or data message with $RESP \leq A$ or $RESP > N$	Ignore
•Receive NAK with $A \leq Resp \leq N$	For all messages with $A < NUM \leq Resp$ inform user of their reception, $A \leftarrow Resp$ $T \leftarrow A+1$ Stop timer

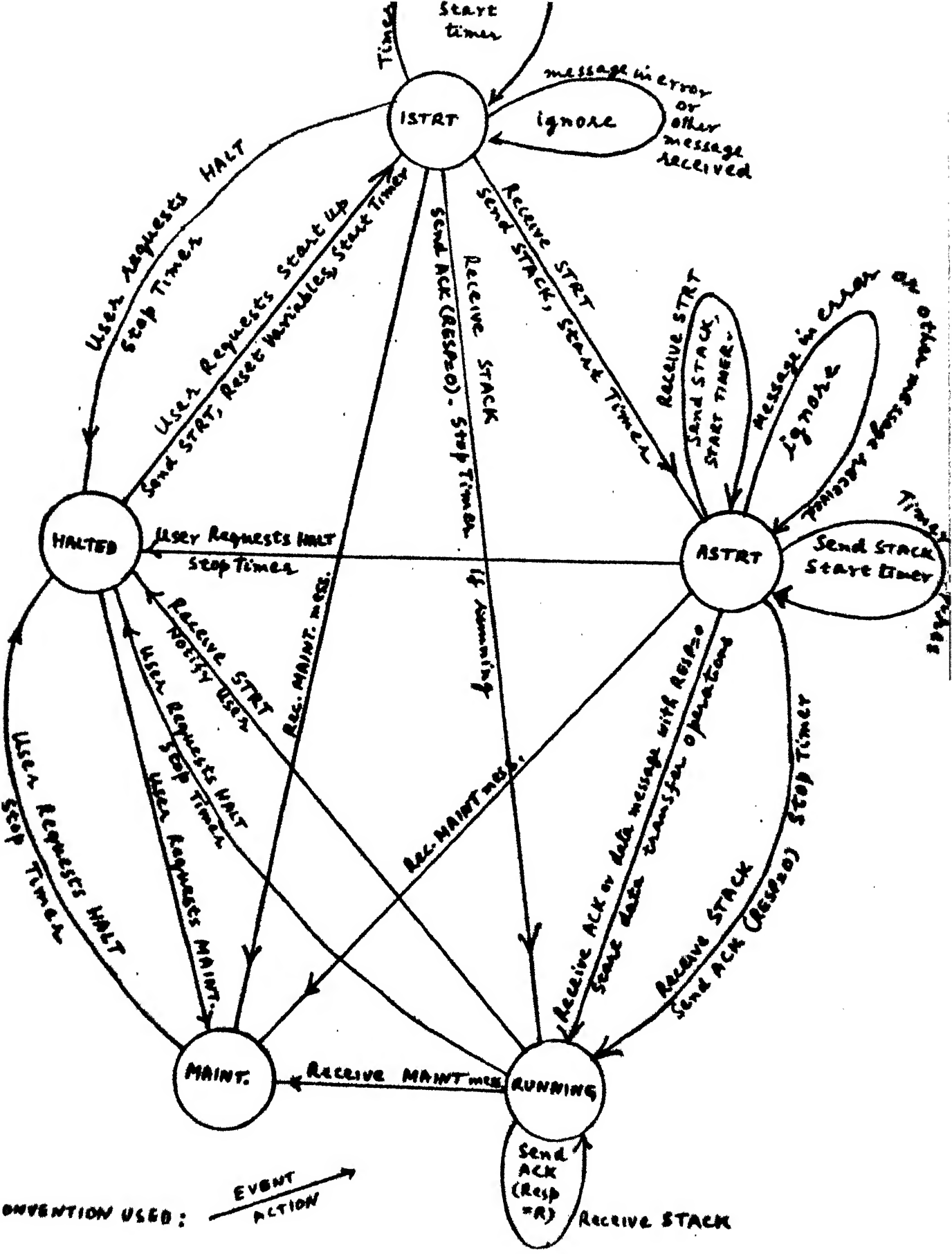


FIG 3.2 : STARTUP STATE DIAGRAM

•Receive NAK with RESP. $< A$ or Resp $> N$	Ignore
•Timer expires	Set SREP flag
•Transmitter idle, SNAK set	Send NAK, Clear SNAK.
•Transmitter idle, SNAK clear, SREP set	Send REP, clear SREP.
•Transmitter idle, SNAK and SREP clear $T < N+1$	Retransmit message T, $T \leftarrow T+1$, Clear SACK.
•User requests message to be sent, T=N+1, Transmitter idle, SNAK and SREP clear	Send message with number N+1, $N \leftarrow N+1$, $T \leftarrow N+1$ Clear SACK
•Transmitter idle, NAK and SREP clear, No user request waiting, SACK set	Send ACK Clear SACK.
•Data Message (NUM) transmission completed on link	$X \leftarrow$ NUM If $A < X$ and timer not running start timer If $A \geq X$, stop timer
•REP message transmission completed on link	Start timer

Note: (1) The term 'send' used above means put the message in a transmitter queue.

(2) The term 'transmitter idle' above means that queuing space for transmission is available.

CHAPTER 4

IMPLEMENTATION OF DDCMP ON MICRO-78

DDCMP was described in detail in Chapter 3. Now, we shall discuss its implementation on MICRO-78. Since the MICRO-78 is linked with three computers, i.e., DEC-1090, IBM-1800 and TDC-316, we require three different implementations of DDCMP on it, one for each link. However, the basic structure of all the three implementations is identical and the differences lie only in some finer details, which will be made clear when we discuss those details. First we shall discuss the basic structure of the implementations.

4-1. BASIC STRUCTURE

There are six basic routines, viz., the transmitter interrupt routine, the receiver interrupt routine, the DDCMP receive routine (DDR), the user transmit routine (USEND) and the user receiver (USRCV) routine. These routines are interfaced by various queues as shown in Figure 4-1. The functions of each of these routines are briefly described below:

(1) USEND: This is the user's routine and is used to pass on messages and commands to DDCMP. Whenever the user wants to send a message or give some command (startup, halt, etc.) he takes a free buffer from FREQ, which is a pool (queue) of free buffers, puts his message/command in that buffer and places it in a queue, Q1.

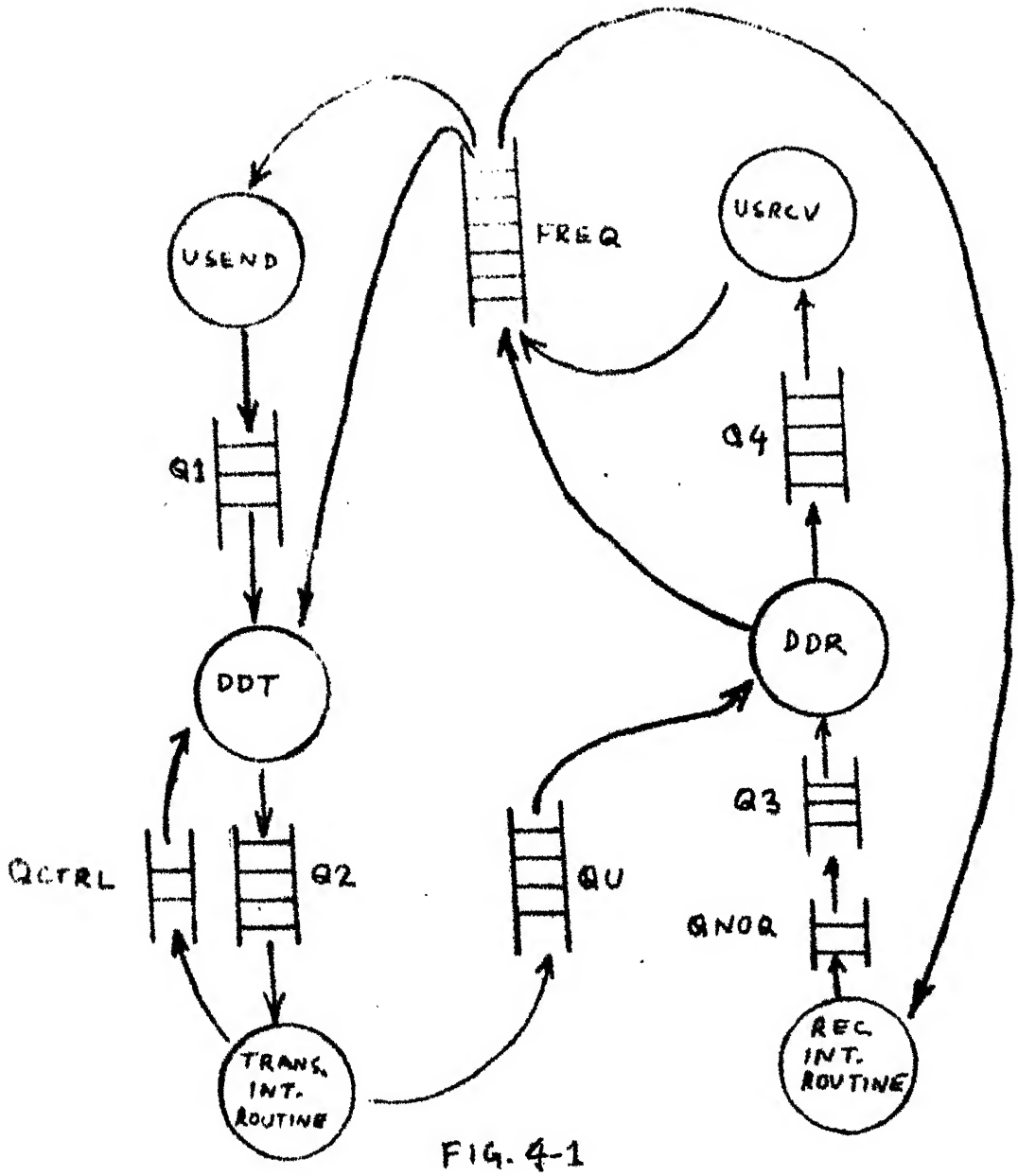


FIG. 4-1

NOTE:

i) THE ARROWS INDICATE VARIOUS PATHS THAT MAY BE TAKEN BY THE BUFFERS WHEN DDCMP IS RUNNING.

ii) QCTRL HAS ATMOST ONE BUFFER IN IT AND IS MEANT TO KEEP A BUFFER FOR SENDING CONTROL MESSAGES.

iii) QNOQ HAS ALWAYS GOT ONE BUFFER FOR RECEIVING A MESSAGE.

(2) DDT: This routine is a component of DDCMP that is responsible for the transmission aspect. This routine decides what has to be transmitted next, i.e., whether a control message a new data message or an old data message should be sent next. It also checks up whether there is any halt or startup message in Q1. Accordingly, it takes appropriate action, i.e., halting the protocol or sending the next message after proper formatting of header, CRC, etc. The messages to be sent are placed in a queue Q2.

(3) Transmitter Interrupt Routine: This routine takes messages one by one from Q2 and sends them on the link. After a complete message has been sent, this routine also takes some other actions like starting a timer, stopping a timer etc. If a control message has been sent, its buffer is returned to ~~FREQ~~ QCTRL, but if a data message has been sent, it is placed in QU, the queue of unacknowledged messages.

(4) Receiver Interrupt Routine: This routine is responsible for receiving messages from the line and placing them in the queue Q3. It basically performs the function of message framing.

(5) DDR: This routine is the receiving component of DDCMP. It takes messages from Q3, examines them, checks their CRC block checks and accordingly takes appropriate actions. The correct data messages are placed in a queue Q4 for the user.

(6) USRCV: This is a user's routine that takes messages one by one from Q4 and uses them appropriately. The freed buffers are sent to FREQ.

4-2. SPECIAL FEATURES OF THE IMPLEMENTATION

Due to some peculiarities of the hardware interfaces, certain special features had to be introduced in the implementation of DDCMP, which are discussed in the following subsections.

4-2.1 Special Features of Implementation on IBM-1800 Side:

(i) Although DDCMP specifies that the receiving end should synchronize only after the reception of two consecutive SYN characters, our interface synchronizes after receiving just one character. Moreover, the value of the SYN character is 026_8 instead of 226_8 as specified by the protocol.

(ii) The hardware interface recognizes the character 001_8 as the starting character of any message and an interrupt is generated when this character is received. However, in DDCMP we have 3 starting characters, SOH(201_8), ENQ(005_8), DLE(220_8). So, we precede every message with the character 001_8. This satisfies the purpose of implementing DDCMP without making any hardware changes.

(iii) We resynchronize after every message. At least 2 SYNs are sent before any message. Consequently the flag QSYNC is not used in the implementation.

(iv) A filler byte (SYN) is sent/received after the first character (SOM) in any message. It is only after the filler byte that the header of a message starts. This is done because on IBM-1800 side, we have got only word operations (one word = 16 bits).

4-2.2 Special Features of Implementation on the TDC-316 Side: There are three special features of the implementation on the TDC-316 side and they are identical to the features (i), (ii) and (iii) described in Section 4-2.1.

4-2.3 Special Features of Implementation on the DEC-10 Side:

(i) On DEC-10 side, the value of the SYN character is 226_8 as against the value 026_8 used on other links. This value conforms to the DDCMP specifications. However, we synchronize after detecting just 1 SYN character.

(ii) The message sent/received on the MICRO to DEC side is a standard DDCMP message, without having any other character (like SOM in other 2 cases) preceding it.

(iii) While sending a message, a sequence of at least 4 SYNs is sent preceding the message. So QSYNC flag is of no concern while sending a message. However, when a message is received, QSYNC is checked and appropriate action taken.

In the following sections we shall discuss the implementation of DDCMP for the MICRO-78 to IBM-1800 link. The implementations for the other two links will be different only in the special features discussed in Section 4-2 and no separate discussion is needed for them.

4-3. BUFFER FORMATTING AND QUEUES

Buffers are arranged as doubly linked lists. Every list has a header which occupies 5 bytes in the format shown in Figure 4-2(a).

The buffers for transmitting or receiving messages have the format shown in Figure 4-2(b). It should be noted that the field

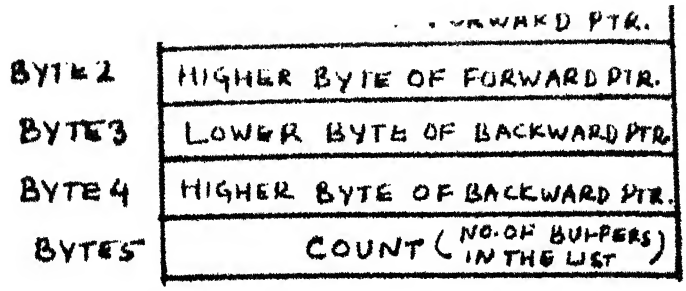


FIG 4.2 (a) : QUEUE HEADER

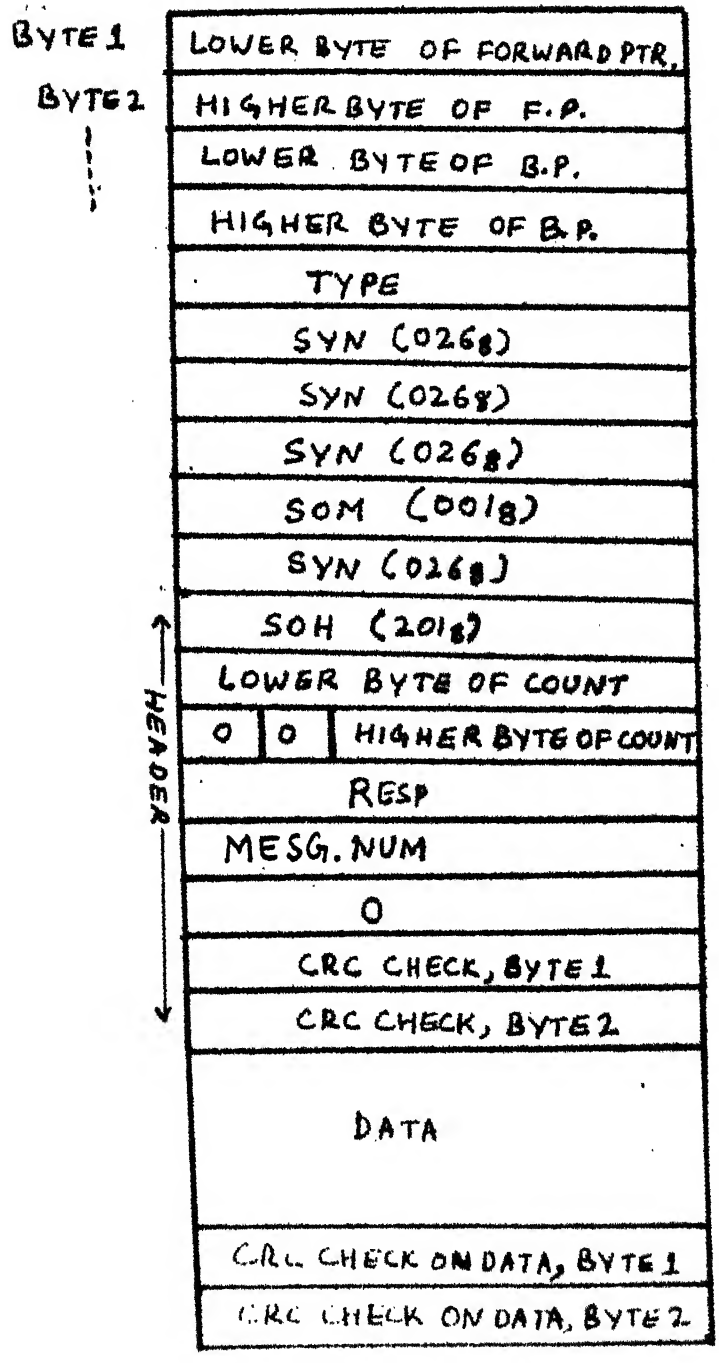


FIG 4.2 (b) : BUFFER FORMAT FOR SENDING/RECEIVING DATA MESSAGES

'message number' is relevant only in the case of data messages. The field 'type' is interpreted as follows:

Value	Meaning
0	Data message
1	ACK message
2	NAK message
3	REP message
6	STRT message (Also serves the purpose of 'start protocol' command from the user)
7	STACK message
99	HALT command (From user to DDCMP)

4-4. DATA VARIABLES AND CONTROL FLAGS

As already described in Chapter 3, various data variables R,T,N,A,X and control flags SACK, SNAK and SREP are required for DDCMP implementation. Besides, another variable indicating the state of DDCMP is also there. This variable (STATE) has got the following interpretation.

Value	State of DDCMP
0	ISTRT
1	ASTRT
2	RUNNING
3	HALTED
4	MAINTENANCE

4-5. ROUTINES FOR QUEUE OPERATIONS

There are four routines for various queue operations. These routines may be called by any other routine which needs to perform operation on the queues. These routines are:

(1) SHFTQ: This routine shifts a buffer from one queue to another, i.e., it removes the first buffer of one queue and appends it to the end of another queue. The address of the header of the queue from which a buffer has to be removed must be placed in H and L registers

and the address of the header of the queue to which this buffer will be shifted must be placed in D and E registers before calling this routine. Its calling sequence is

CALL SHFTQ

SHFTQ should be called only when the queue from which a buffer is to be shifted is non-empty, otherwise a fatal error is assumed to have occurred and the protocol halts after printing an error message (Ref. Appendix I).

(2) SRCHQ: This routine searches a list for a buffer with a particular message number (i.e., a particular value in the NUM field. Ref. Figure 4-2(b)) and after finding that buffer, removes it from the original list and places it at the end of another list. It is essential that the address of the header of the queue in which the buffer is to be searched and removed be kept in the H and L registers the address of the header of the queue to which this buffer will be shifted to be placed in D and E registers and the number of the message be kept in the location called 'SNUM' before the routine is called. Its calling sequence is

CALL SRCHQ.

If the search succeeds, a flag SRERR is set, otherwise it is reset. The calling routine should examine SRERR and take appropriate action.

(3) EMPTQ: This routine shifts all the buffers of one queue to another queue. The address of the header of the queue from which all the buffers have to be removed must be placed in H and L register

and the address of the header of the queue to which all these buffers will be shifted to must be placed in D and E registers before calling this routine. Its calling sequence is

CALL EMPTQ

The queue from which all the buffers have to be shifted out may be empty.

(4) SRCQ2: This routine searches a list for a buffer with a particular message number and after finding that buffer, sets the SRERR flag and if the search fails this flag is reset. The address of the header of the queue in which this buffer is looked for must be placed in H and L registers and the message number must be placed in SNUM location before calling this routine. Its calling sequence is

CALL SRCQ2.

4-6. DETAILED EXPLANATION OF VARIOUS ROUTINES

(1) USEND: This routine takes an empty buffer from FREQ, and places it in a temporary queue TEMQ* and fills it as follows:

(i) If the user wants to give a command which will halt the protocol after sending all the previously given messages he places number 99 in the 'type' field.

(ii) If the user wants to give a start up command, he places No. 6 in the type' field.

---

*TEMQ is another queue which is used to hold a buffer temporarily while it is being processed by some routine. This queue never contains more than one buffer.

(iii) If the user wants to send a data message, he places no. '0' in the type field, the number of data bytes in the COUNT field and the data in the data field. After the buffer has been filled with relevant information it is shifted from TEMQ to Q1.

If the user wishes to halt the protocol immediately (i.e., even before the previous messages are transmitted), he simply sets a flag called SHALT.

(2) DDT: The basic function of this routine has already been explained. The details are clear from the flowchart given in Figure 4-3(a). The following points will help in understanding the flowcharts.

(i) There are six header routines, MES, ACK, NAK, REP, STRT and STACK (Figures 4-4(a) to (f)), which generate the header for that message type. These routines fill in all the information shown in the transmitting side buffers (Figure 4-2(b)) except the data in data messages and the CRC check on the data. These routines need only one parameter to be passed, i.e., the address of the header of that queue which contains the buffer to be filled in at the front.

(ii) All arithmetic and comparison operations shown are modulo 256.

(iii) A routine for CRC generation is called at various points. Before calling this routine, the address of the first location containing the data on which CRC is to be generated/checked should be placed in registers H and L, the count of data bytes should be placed in registers B and C and a location FLAG should be set to 0 or 1 depending on whether CRC generation or checking is required. When this routine is called for CRC generation (FLAG = 0) two CRC check bytes are placed in two locations immediately following the data. When it

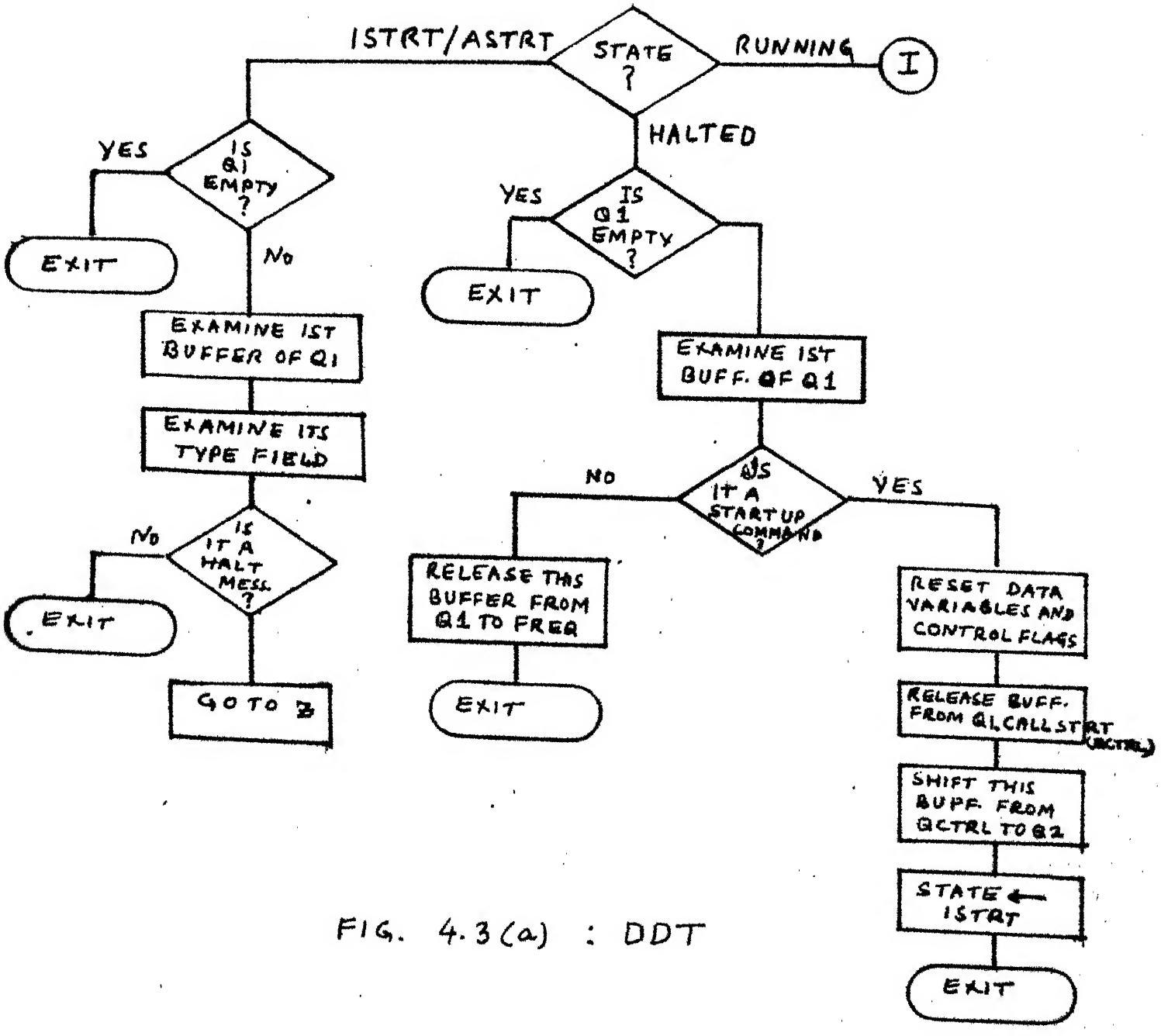


FIG. 4.3(a) : DDT

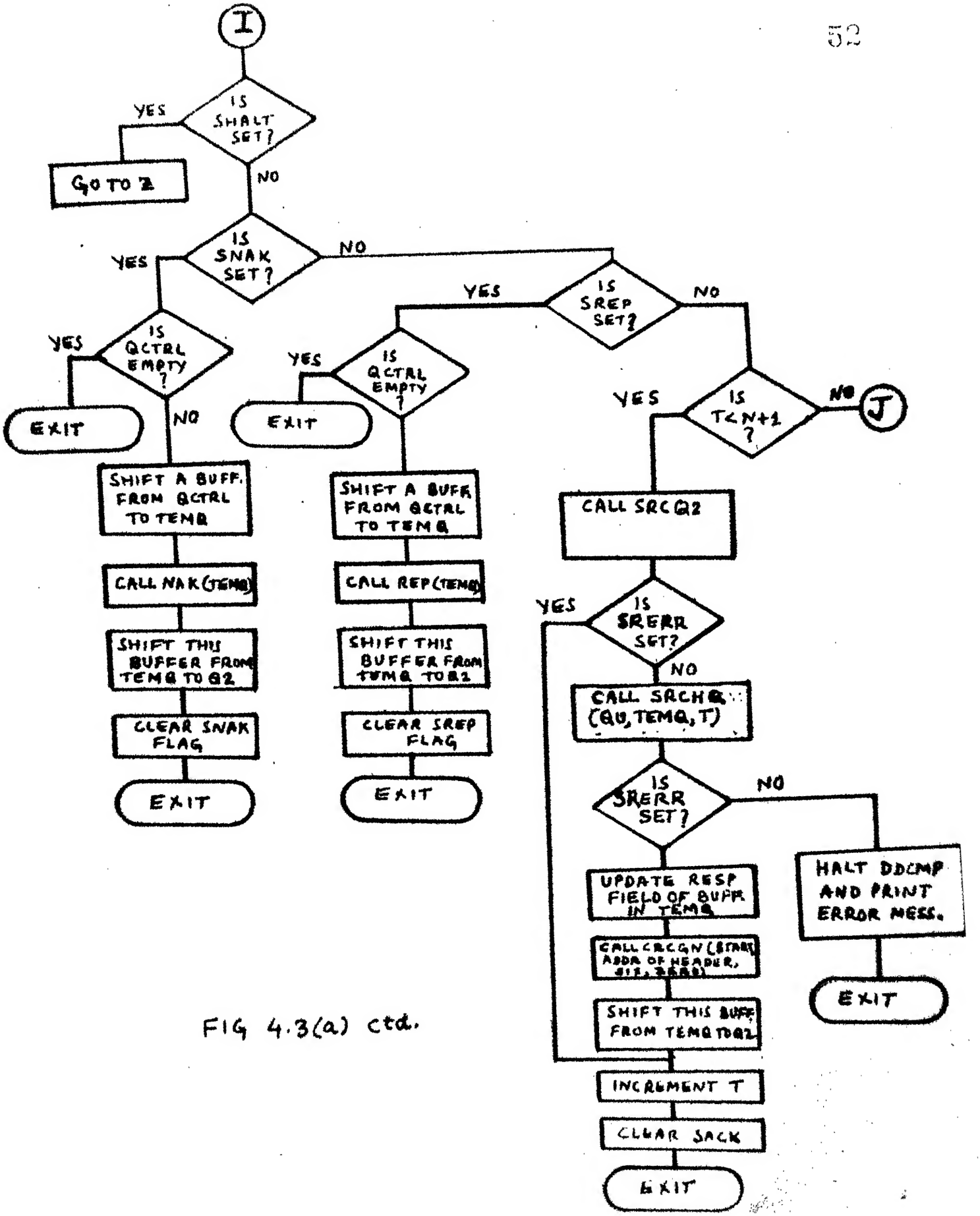

I
IS SHALT SET?
YES
GO TO 2
NO
IS SNAK SET?
YES
NO
IS QCTRL EMPTY?
YES
EXIT
NO
SHIFT A BUFF. FROM QCTRL TO TEMQ
CALL NAK(TEMQ)
SHIFT THIS BUFFER FROM TEMQ TO Q2
CLEAR SNAK FLAG
EXIT
IS SREP SET?
YES
NO
IS QCTRL EMPTY?
YES
EXIT
SHIFT A BUFF FROM QCTRL TO TEMQ
CALL REP(TEMQ)
SHIFT THIS BUFFER FROM TEMQ TO Q1
CLEAR SREP FLAG
EXIT
IS T<N+1?
YES
NO
J
CALL SRCQ2
IS SRERR SET?
YES
NO
CALL SRCHQ (QU, TEMQ, T)
IS SRERR SET?
NO
UPDATE RESP FIELD OF BUFF IN TEMQ
CALL CRCGN (START ADDR OF HEADER, SIZE, ZERO)
SHIFT THIS BUFF FROM TEMQ TO Q2
INCREMENT T
CLEAR SACK
EXIT
HALT DDCMP AND PRINT ERROR MESS.
EXIT

FIG 4.3(a) ctd.

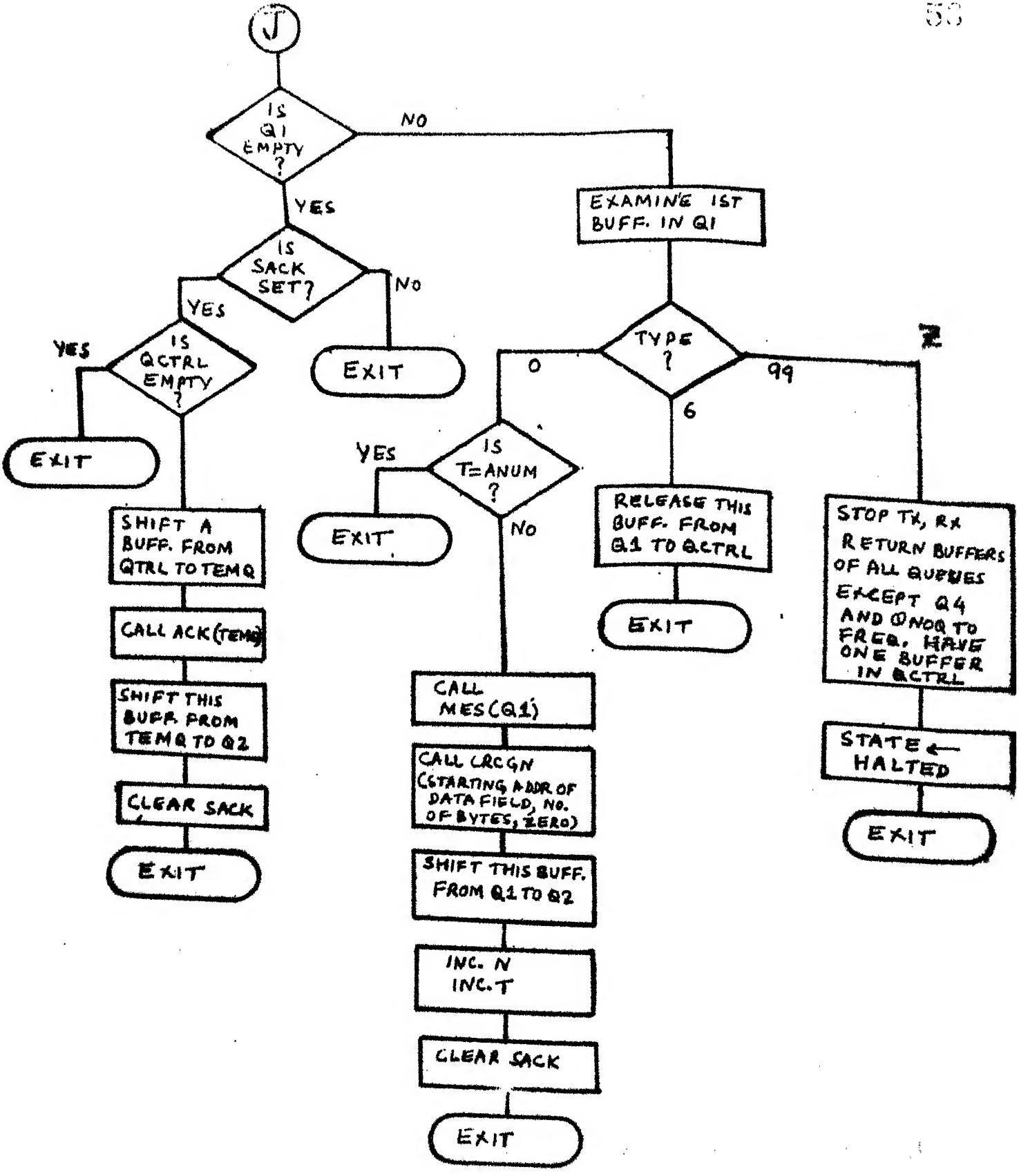
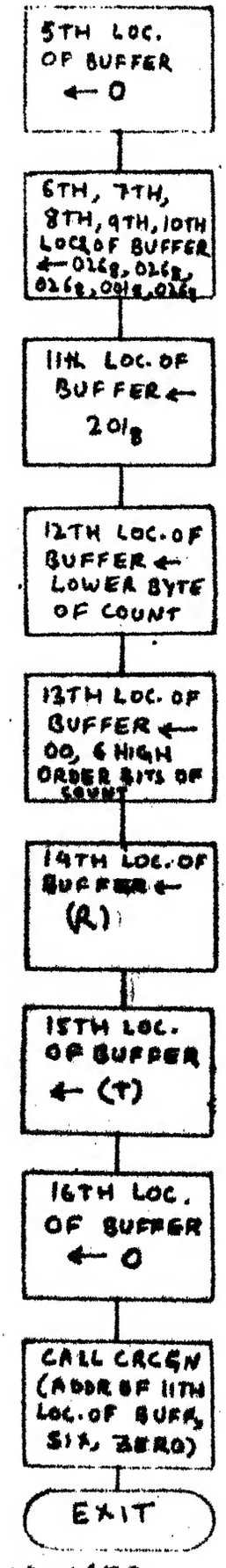
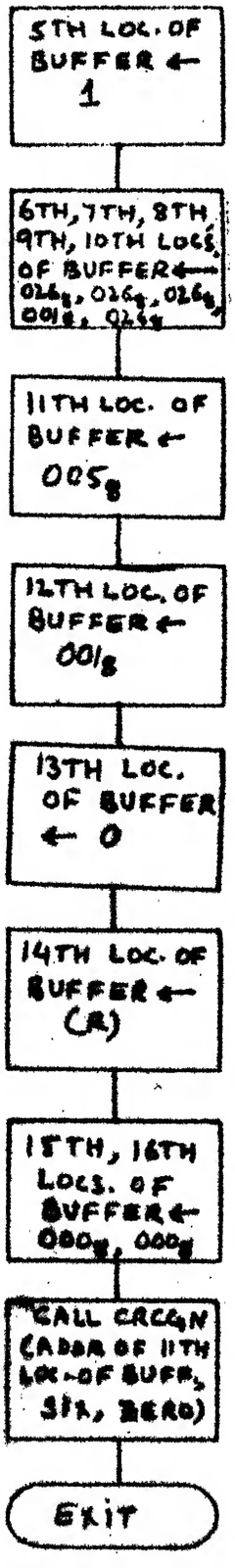

J
IS Q1 EMPTY ?
NO
YES
IS SACK SET?
NO
YES
EXIT
IS QCTRL EMPTY ?
YES
EXIT
SHIFT A BUFF. FROM QTRL TO TEMQ
CALL ACK(TEMQ)
SHIFT THIS BUFF. FROM TEMQ TO Q2
CLEAR SACK
EXIT
EXAMINE 1ST BUFF. IN Q1
TYPE ?
0
6
99
IS T=ANUM ?
YES
EXIT
NO
CALL MES(Q1)
CALL CRCGN (STARTING ADDR OF DATA FIELD, NO. OF BYTES, ZERO)
SHIFT THIS BUFF. FROM Q1 TO Q2
INC. N
INC. T
CLEAR SACK
EXIT
RELEASE THIS BUFF. FROM Q1 TO QCTRL
EXIT
STOP TX, RX
RETURN BUFFERS OF ALL QUEUES EXCEPT Q4 AND QNOQ TO FREQ. HAVE ONE BUFFER IN QCTRL
STATE ← HALTED
EXIT

FIG 4.3(a) ctd.


5TH LOC. OF BUFFER ← 0
6TH, 7TH, 8TH, 9TH, 10TH LOCS OF BUFFER ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 201₈
12TH LOC. OF BUFFER ← LOWER BYTE OF COUNT
13TH LOC. OF BUFFER ← 00, 6 HIGH ORDER BITS OF COUNT
14TH LOC. OF BUFFER ← (R)
15TH LOC. OF BUFFER ← (T)
16TH LOC. OF BUFFER ← 0
CALL CRC&N (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
a) MES


5TH LOC. OF BUFFER ← 1
6TH, 7TH, 8TH 9TH, 10TH LOCS, OF BUFFER ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 005₈
12TH LOC. OF BUFFER ← 001₈
13TH LOC. OF BUFFER ← 0
14TH LOC. OF BUFFER ← (R)
15TH, 16TH LOCS. OF BUFFER ← 000₈, 000₈
CALL CRC&N (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
b) ACK

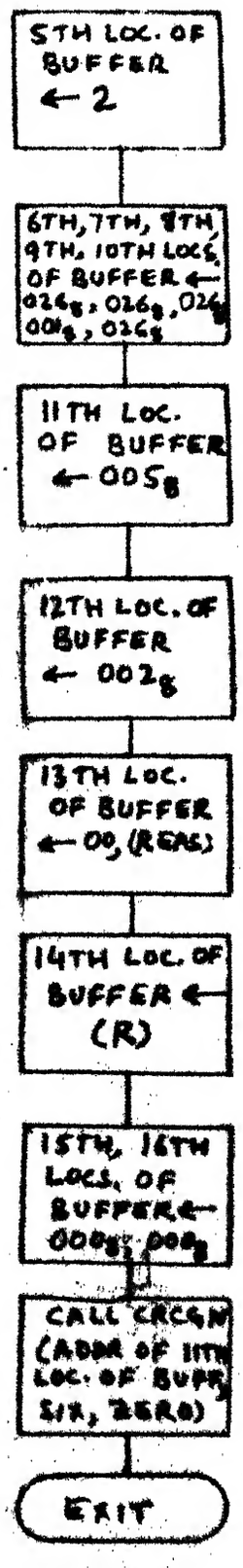

5TH LOC. OF BUFFER ← 2
6TH, 7TH, 8TH 9TH, 10TH LOCS, OF BUFFER ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 005₈
12TH LOC. OF BUFFER ← 002₈
13TH LOC. OF BUFFER ← 00, (REAS)
14TH LOC. OF BUFFER ← (R)
15TH, 16TH LOCS. OF BUFFER ← 000₈, 000₈
CALL CRC&N (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
c) NAK

FIG. 4.4

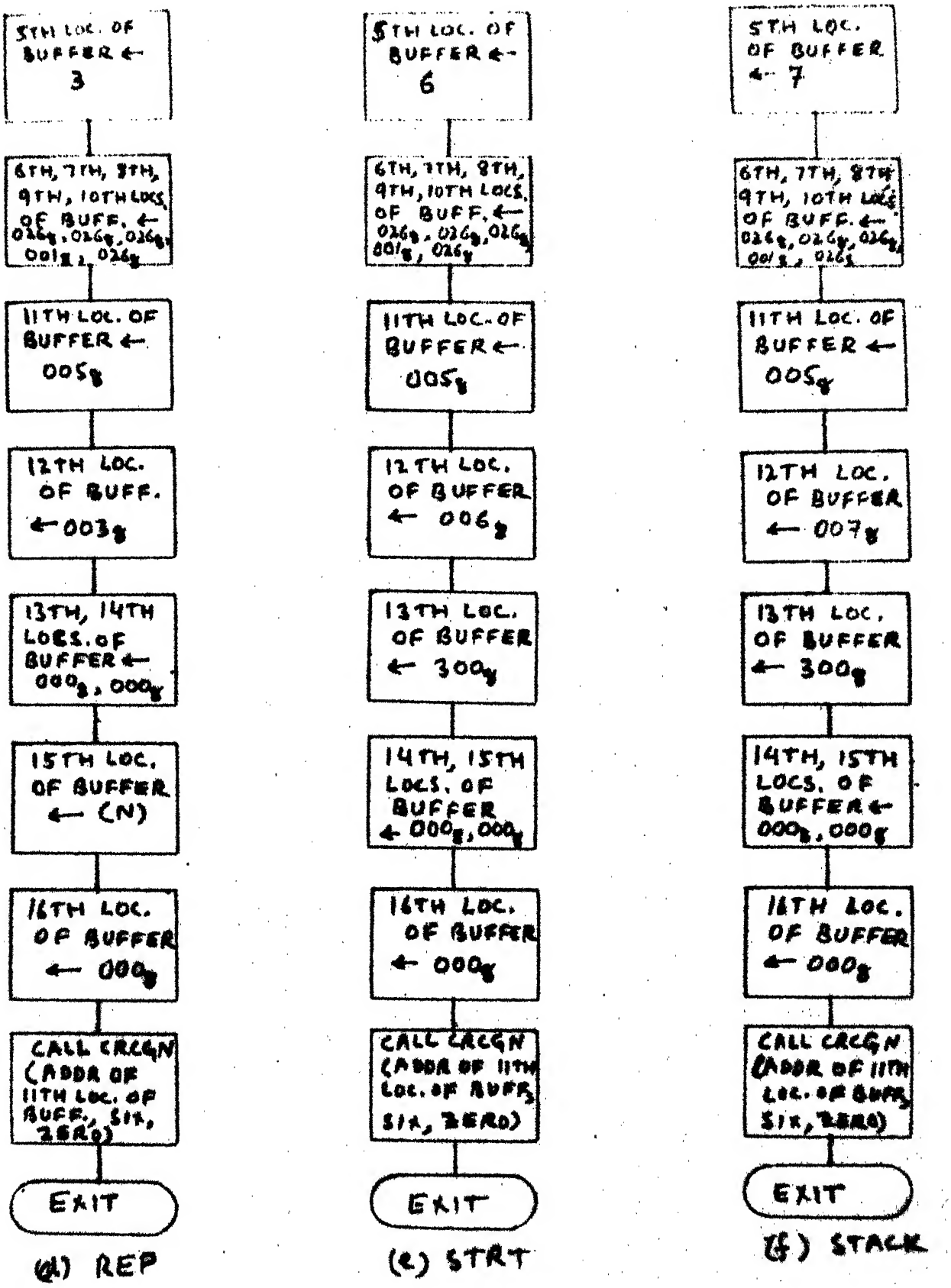

5TH LOC. OF BUFFER ← 3
6TH, 7TH, 8TH, 9TH, 10TH LOCS. OF BUFF. ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 005₈
12TH LOC. OF BUFF. ← 003₈
13TH, 14TH LOCS. OF BUFFER ← 000₈, 000₈
15TH LOC. OF BUFFER ← (N)
16TH LOC. OF BUFFER ← 000₈
CALL CRCGN (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
(d) REP
5TH LOC. OF BUFFER ← 6
6TH, 7TH, 8TH, 9TH, 10TH LOCS. OF BUFF. ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 005₈
12TH LOC. OF BUFFER ← 006₈
13TH LOC. OF BUFFER ← 300₈
14TH, 15TH LOCS. OF BUFFER ← 000₈, 000₈
16TH LOC. OF BUFFER ← 000₈
CALL CRCGN (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
(e) STRT
5TH LOC. OF BUFFER ← 7
6TH, 7TH, 8TH, 9TH, 10TH LOCS. OF BUFF. ← 026₈, 026₈, 026₈, 001₈, 026₈
11TH LOC. OF BUFFER ← 005₈
12TH LOC. OF BUFFER ← 007₈
13TH LOC. OF BUFFER ← 300₈
14TH, 15TH LOCS. OF BUFFER ← 000₈, 000₈
16TH LOC. OF BUFFER ← 000₈
CALL CRCGN (ADDR OF 11TH LOC. OF BUFF, SIX, ZERO)
EXIT
(f) STACK

FIG. 4.4 (Ctd.)

is called for CRC check (FLAG = 1), the variable FLAG is set to 0 if no error is detected, otherwise it remains 1.

(3) Transmitter Interrupt Routine: An interrupt is generated after the transmission of every byte of a message. The interrupt routine initiates the transmission of the next byte, if any. When all the bytes of a message have been transmitted, certain actions are performed as are clear from the flowchart given in Figure 4-5(a). However, this routine has not been coded and tested because the hardware interfaces are still being fabricated.

(4) Receiver Interrupt Routine: When an SOM character is detected by the interface in idle mode, a hardware mode bit is set and an interrupt is generated. For every subsequent byte received while the mode bit is on, an interrupt is generated. This continues till the entire message has been received and after that the mode bit is reset by the interrupt routine. The functions performed by the interrupt routine are clear from the flowchart in Figure 4-5(b). This routine has also not been coded and tested.

(5) DDR: The flowcharts of this routine and those of ACKR and NAKR (Figures 4-6(a) to (c)) explain the functions performed by DDR.

(6) USRCV: This routine takes a message from Q4, makes it available to the user for processing and returns the freed buffer to FREQ.

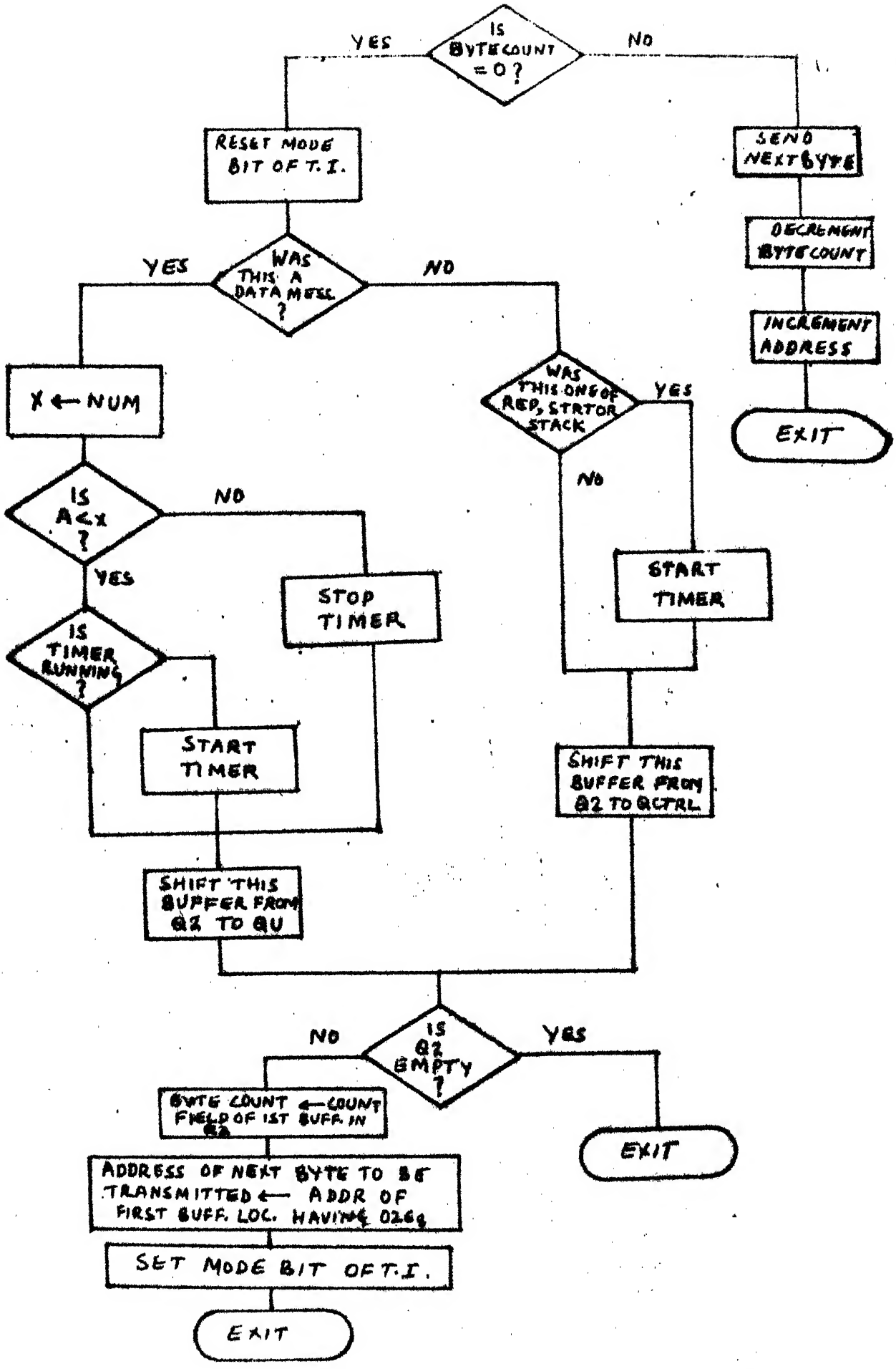


FIG. 4.5(a): TRANSMITTER INTERRUPT ROUTINE

REC : NO. OF BYTES RECD.
RCOUNT: NO. OF BYTES TO BE RECD.

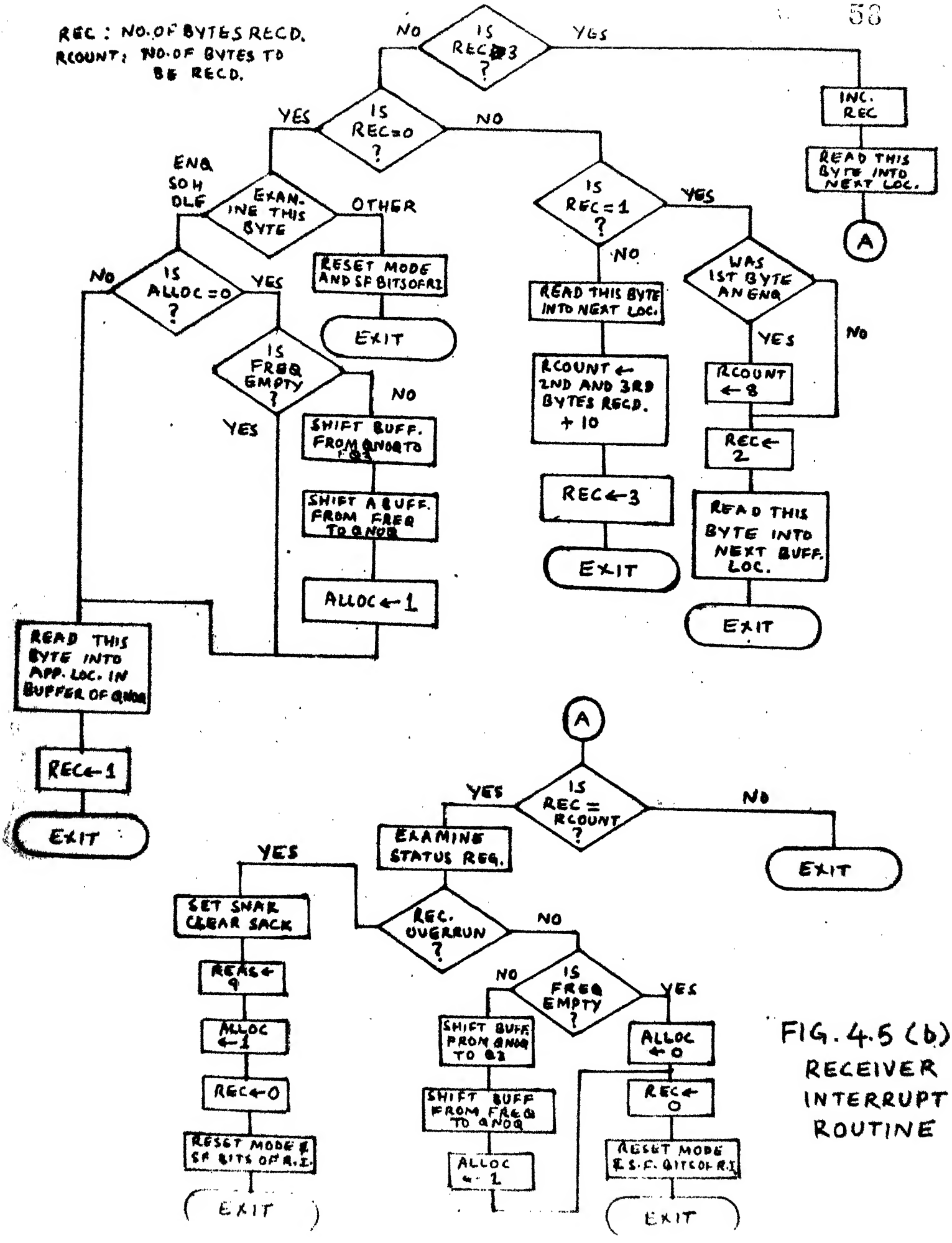


FIG. 4.5 (b): RECEIVER INTERRUPT ROUTINE

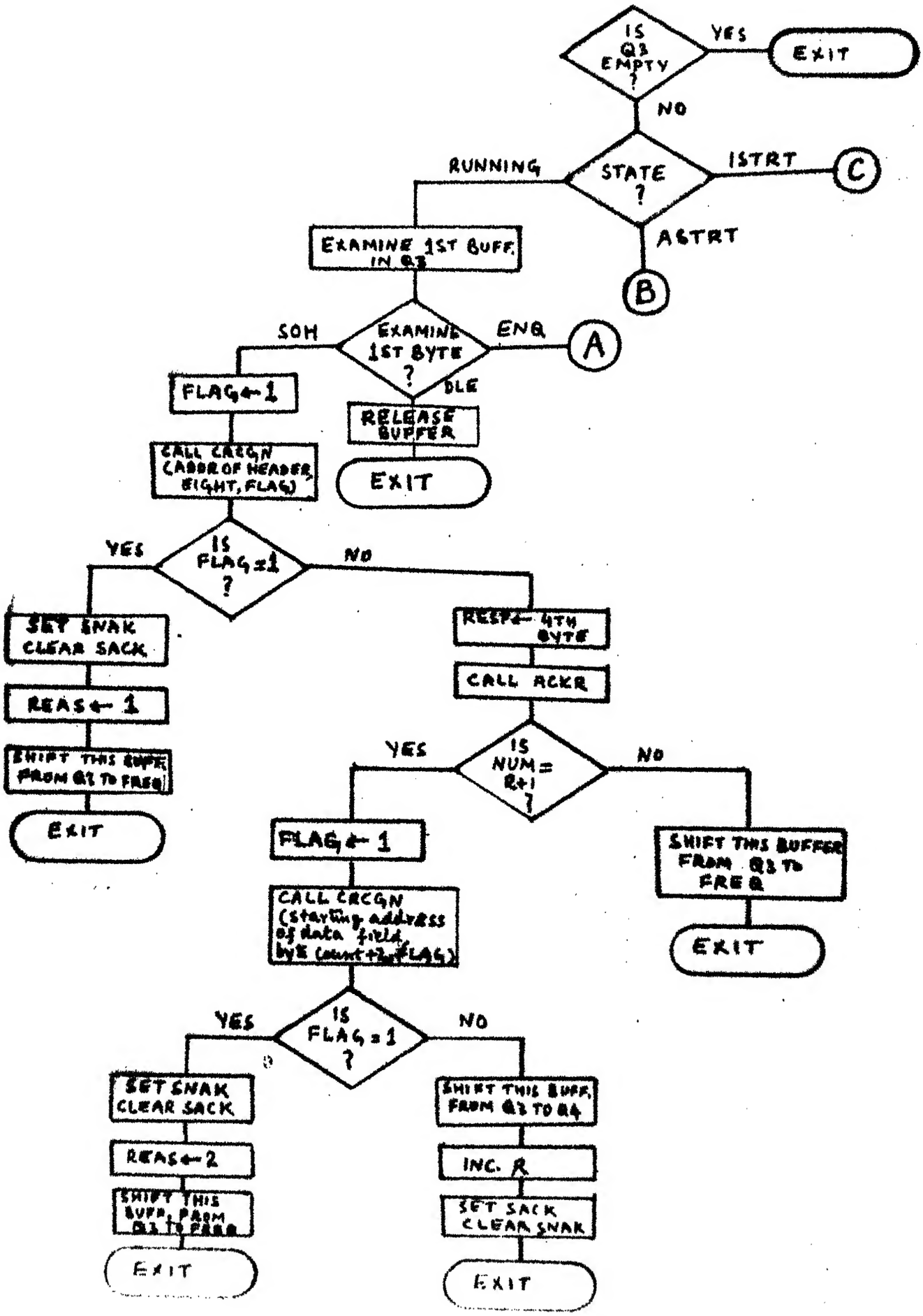

IS Q3 EMPTY ?
YES
EXIT
NO
STATE ?
RUNNING
ISTRT
C
ASTRT
B
EXAMINE 1ST BUFF. IN Q3
EXAMINE 1ST BYTE ?
SOH
ENQ
A
DLE
FLAG ← 1
RELEASE BUFFER
EXIT
CALL CRCGN (ADDR OF HEADER, EIGHT, FLAG)
IS FLAG = 1 ?
YES
NO
SET SNAK CLEAR SACK
REAS ← 1
SHIFT THIS BUFF. FROM Q3 TO FREE Q
EXIT
RESP# ← 4TH BYTE
CALL ACKR
IS NUM = R+1 ?
YES
NO
FLAG ← 1
CALL CRCGN (starting address of data field, byte count+2, FLAG)
SHIFT THIS BUFFER FROM Q3 TO FREE Q
EXIT
IS FLAG = 1 ?
YES
NO
SET SNAK CLEAR SACK
REAS ← 2
SHIFT THIS BUFF. FROM Q3 TO FREE Q
EXIT
SHIFT THIS BUFF. FROM Q3 TO Q4
INC. R
SET SACK CLEAR SNAK
EXIT

FIG 4.6 (a)

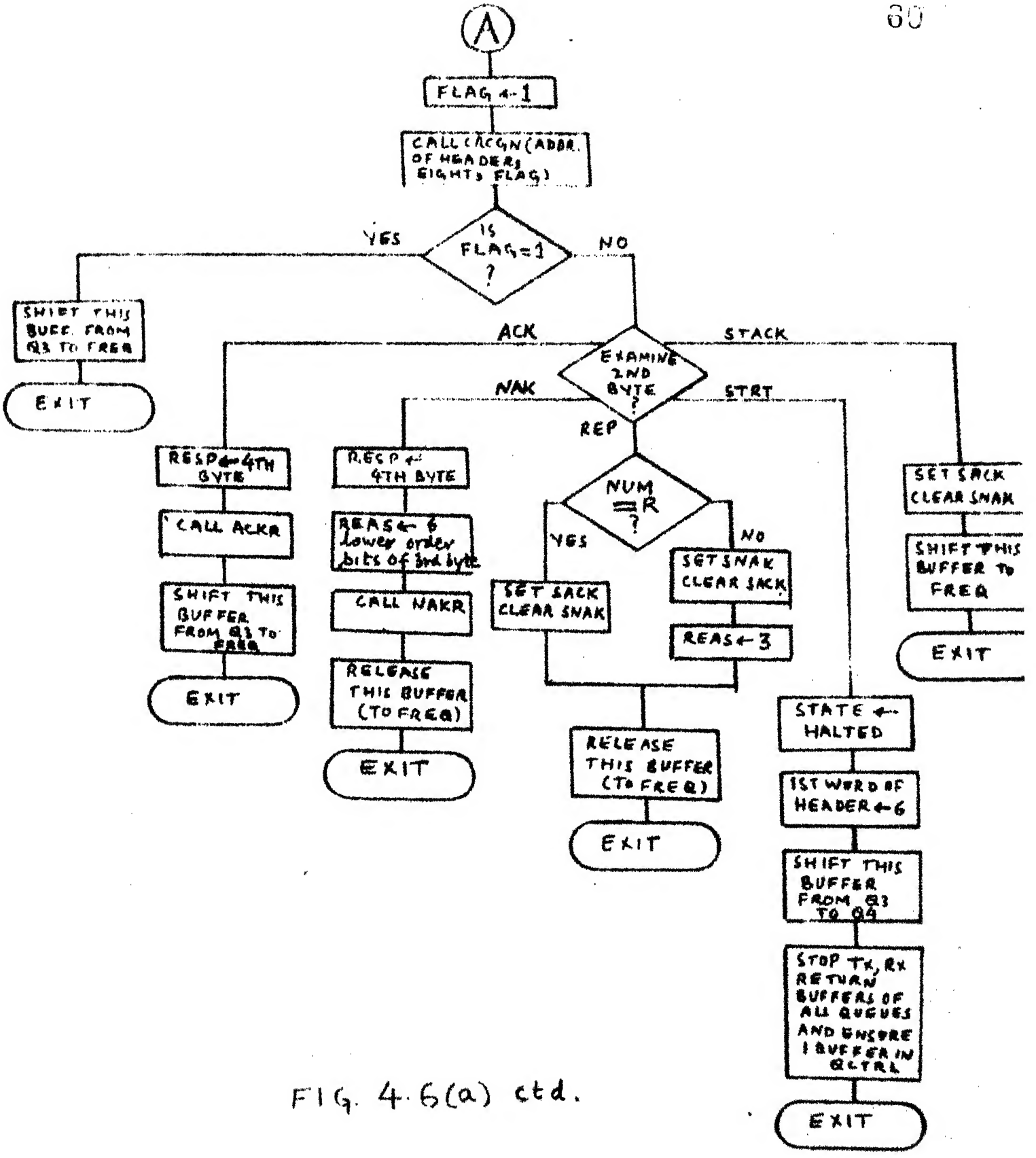

A
FLAG ← 1
CALL (ACQN (ADDR. OF HEADER, EIGHT, FLAG)
IS FLAG = 1 ?
YES
NO
SHIFT THIS BUFF. FROM Q3 TO FREQ
EXIT
ACK
NAK
REP
STRT
STACK
EXAMINE 2ND BYTE ?
RESP ← 4TH BYTE
CALL ACKR
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT
RESP ← 4TH BYTE
REAS ← 6 lower order bits of 3rd byte
CALL NAKR
RELEASE THIS BUFFER (TO FREQ)
EXIT
NUM = R ?
YES
NO
SET SACK CLEAR SNAK
SET SNAK CLEAR SACK
REAS ← 3
RELEASE THIS BUFFER (TO FREQ)
EXIT
SET SACK CLEAR SNAK
SHIFT THIS BUFFER TO FREQ
EXIT
STATE ← HALTED
1ST WORD OF HEADER ← 6
SHIFT THIS BUFFER FROM Q3 TO Q4
STOP TX, RX RETURN BUFFERS OF ALL QUEUES AND ENSURE 1 BUFFER IN Q CTRL
EXIT

FIG. 4.6(a) ctd.

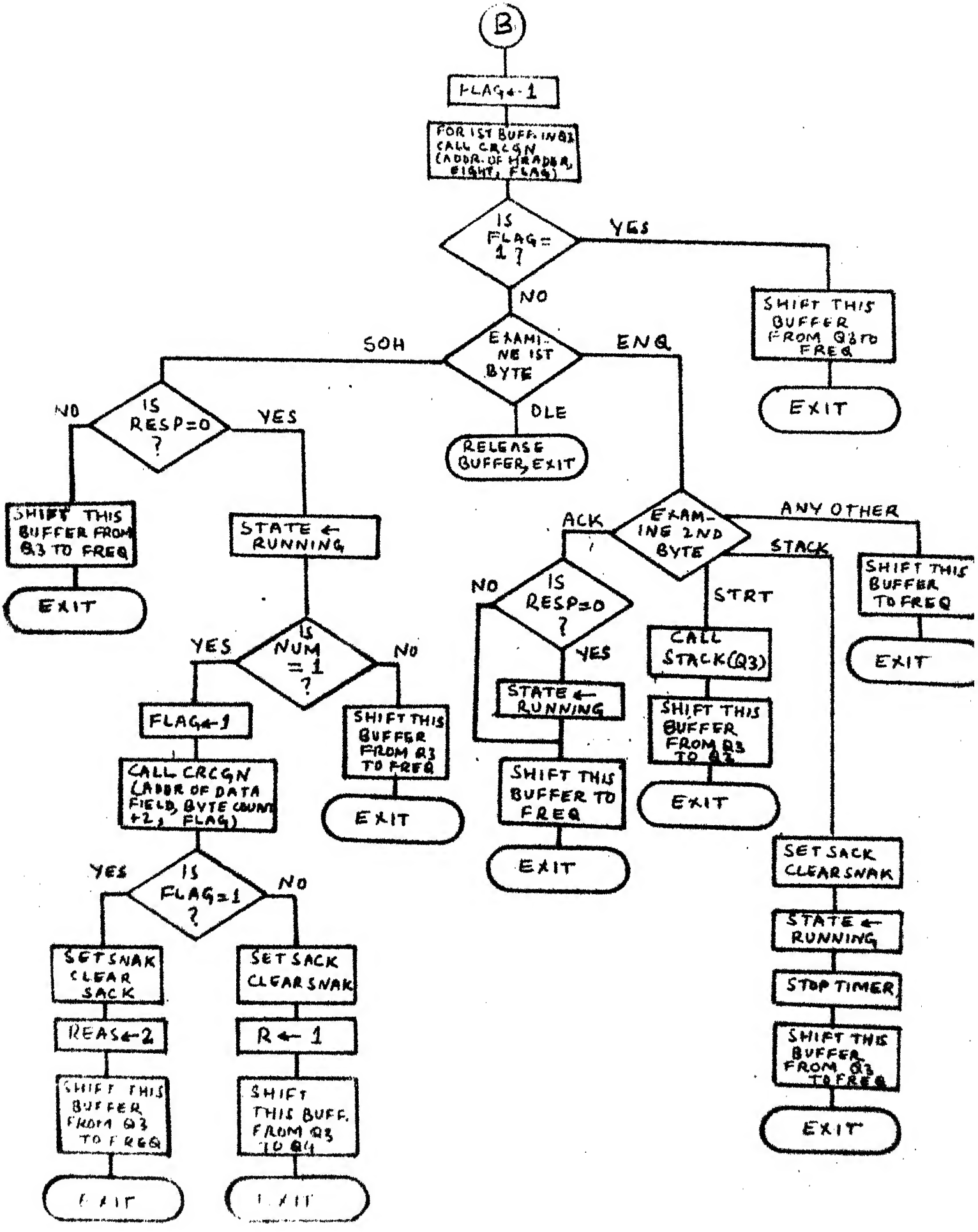

B
FLAG←1
FOR 1ST BUFF. IN Q2 CALL CRCGN (ADDR. OF HEADER, EIGHT, FLAG)
IS FLAG = 1 ?
YES
NO
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT
EXAMINE 1ST BYTE
SOH
ENQ
DLE
RELEASE BUFFER, EXIT
IS RESP=0 ?
NO
YES
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT
STATE ← RUNNING
IS NUM = 1 ?
YES
NO
FLAG←1
CALL CRCGN (ADDR OF DATA FIELD, BYTE COUNT +2, FLAG)
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT
IS FLAG=1 ?
YES
NO
SET SNAK CLEAR SACK
REAS←2
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT
SET SACK CLEAR SNAK
R ← 1
SHIFT THIS BUFF. FROM Q3 TO Q4
EXIT
EXAMINING 2ND BYTE
ACK
STACK
STRT
ANY OTHER
IS RESP=0 ?
NO
YES
STATE ← RUNNING
SHIFT THIS BUFFER TO FREQ
EXIT
CALL STACK(Q3)
SHIFT THIS BUFFER FROM Q3 TO Q2
EXIT
SHIFT THIS BUFFER TO FREQ
EXIT
SET SACK CLEAR SNAK
STATE ← RUNNING
STOP TIMER
SHIFT THIS BUFFER FROM Q3 TO FREQ
EXIT

FIG. 4.6 (a) ctd.

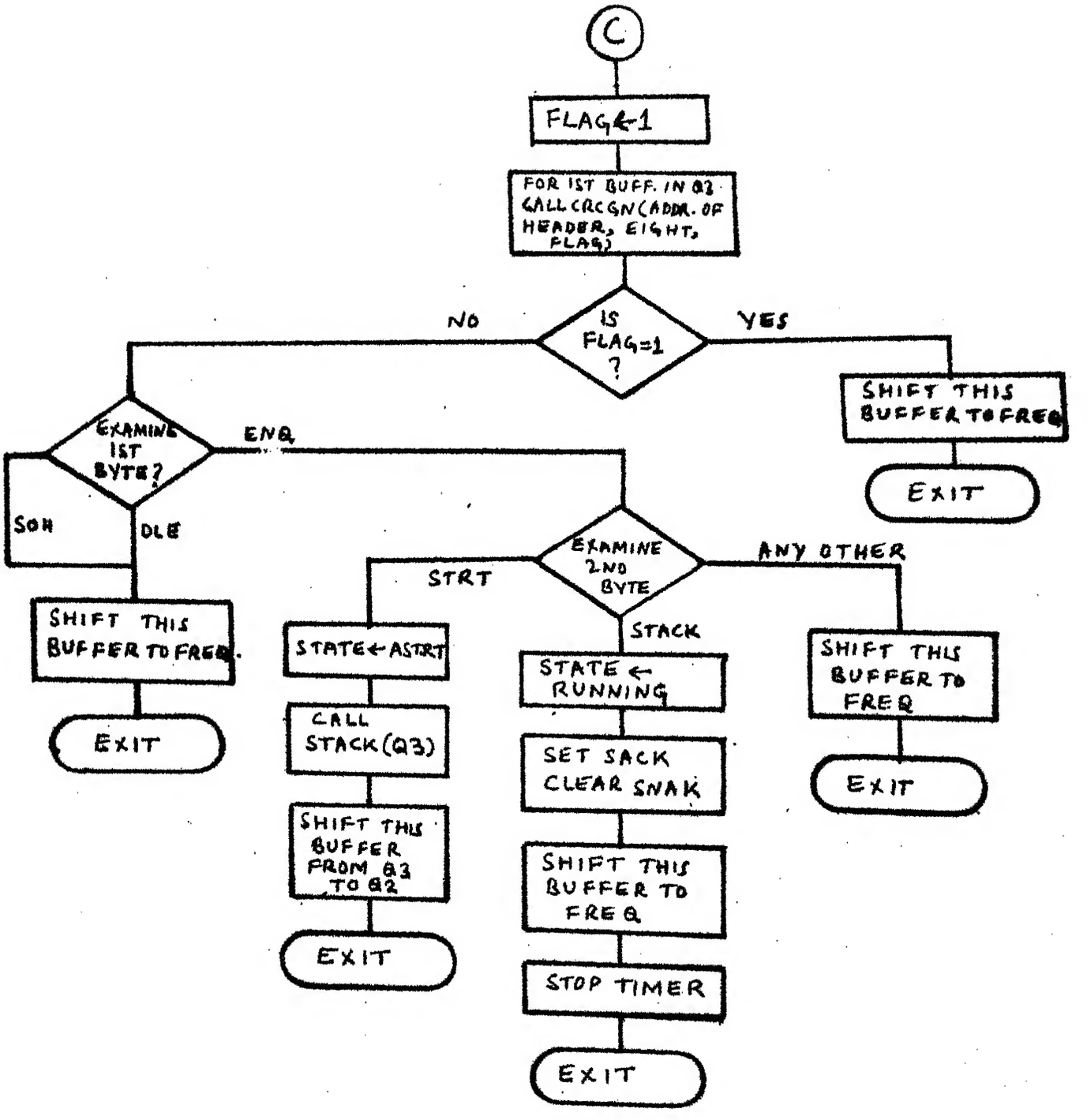

C
FLAG←1
FOR 1ST BUFF. IN Q2 CALL CRCGN (ADDR. OF HEADER, EIGHT, FLAG)
IS FLAG=1 ?
NO
YES
SHIFT THIS BUFFER TO FREE
EXIT
EXAMINE 1ST BYTE?
ENQ
SOH
DLE
SHIFT THIS BUFFER TO FREE
EXIT
EXAMINE 2ND BYTE
STRT
ANY OTHER
STACK
STATE←ASTRT
CALL STACK(Q3)
SHIFT THIS BUFFER FROM Q3 TO Q2
EXIT
STATE ← RUNNING
SET SACK CLEAR SNAK
SHIFT THIS BUFFER TO FREE
STOP TIMER
EXIT
SHIFT THIS BUFFER TO FREE
EXIT

FIG. 4.6 (a) Ctd.

ACKR

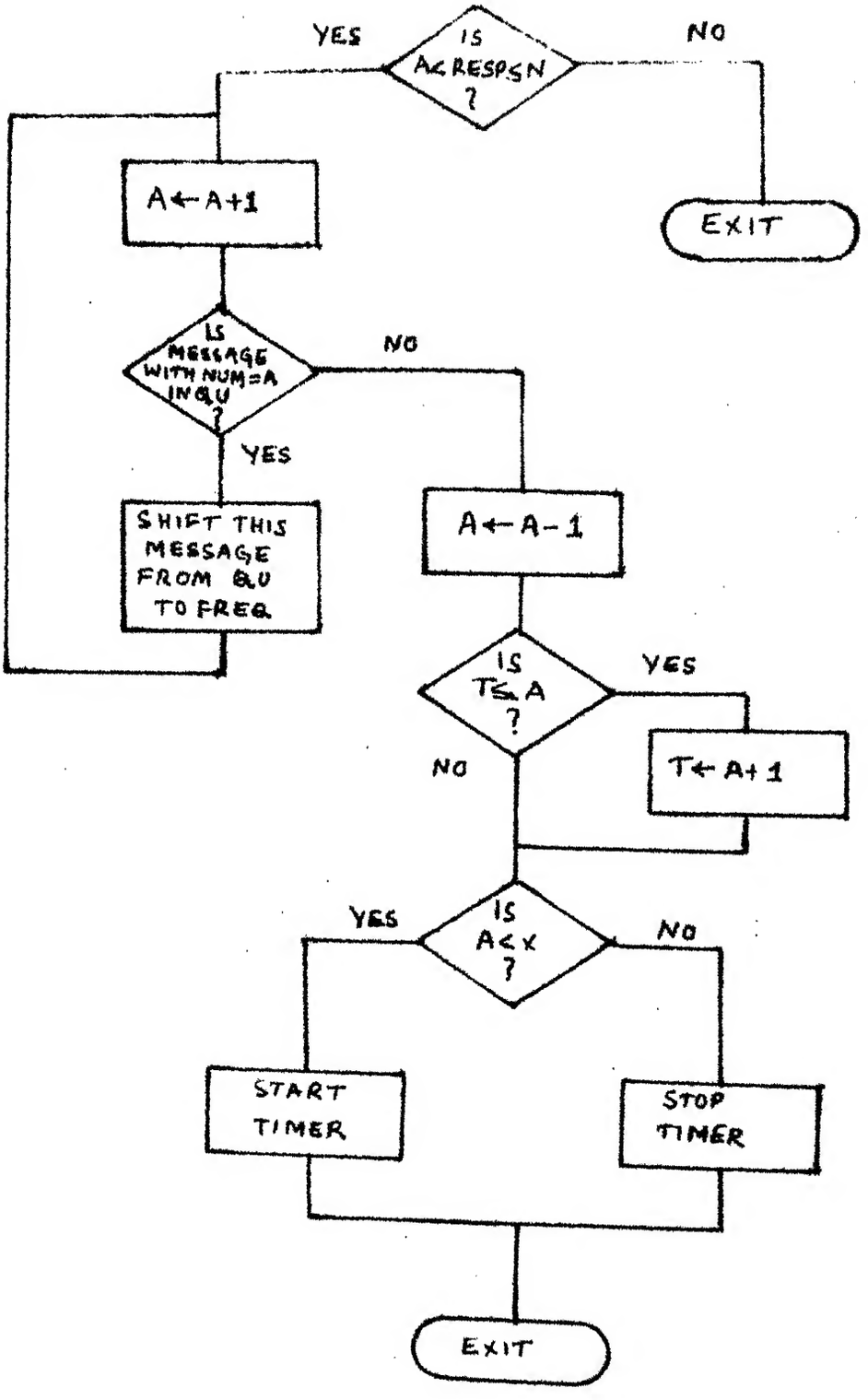

IS AC RESPSN ?
YES
NO
EXIT
A←A+1
IS MESSAGE WITH NUM=A IN &U ?
NO
YES
SHIFT THIS MESSAGE FROM &U TO FREE
A←A-1
IS T≤A ?
YES
T←A+1
NO
IS A<X ?
YES
NO
START TIMER
STOP TIMER
EXIT

FIG. 4.6 (b)

NAKR

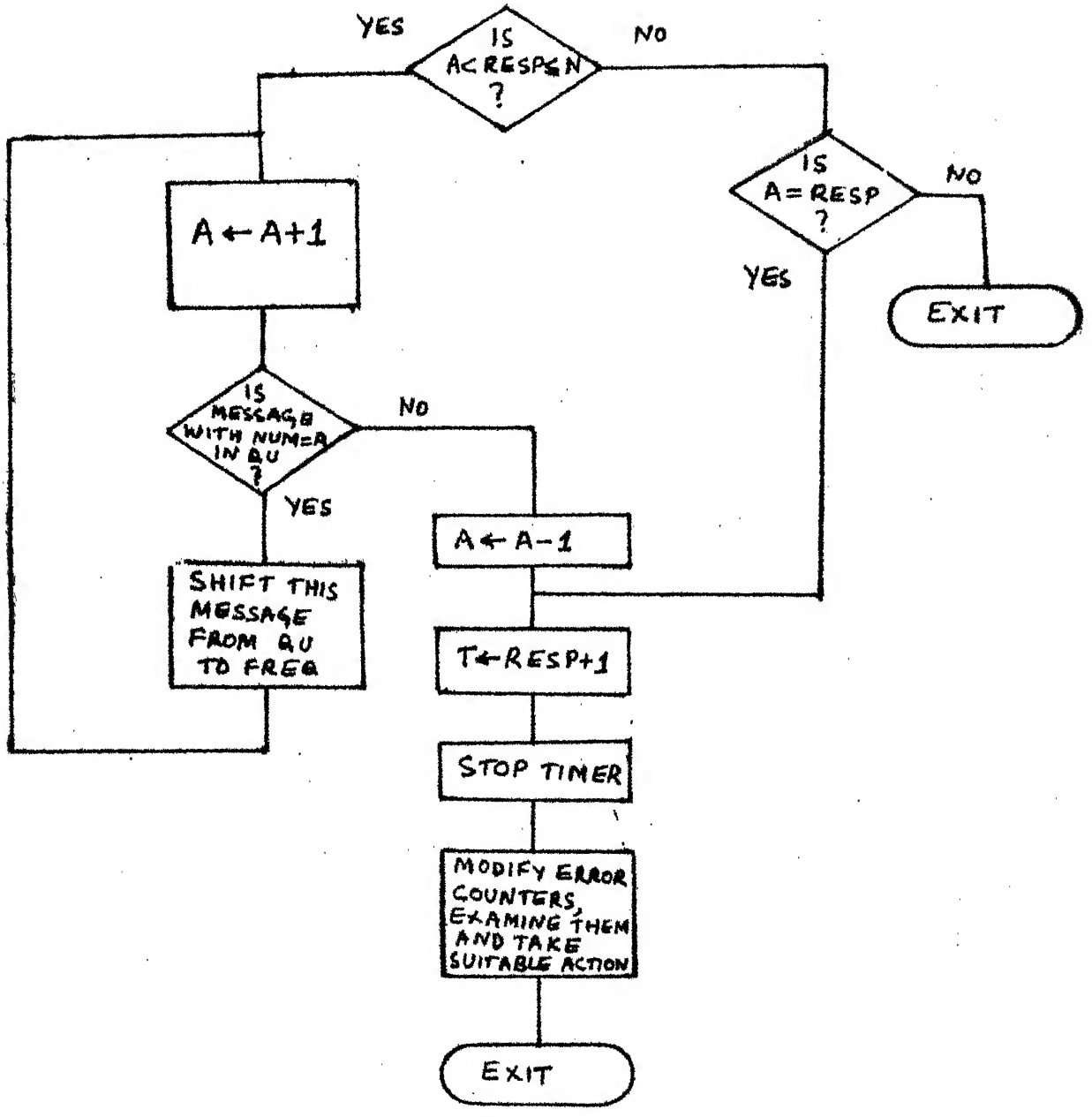

IS A<RESP≤N ?
YES
NO
IS A=RESP ?
NO
YES
EXIT
A←A+1
IS MESSAGE WITH NUM=A IN QU ?
NO
YES
A←A-1
SHIFT THIS MESSAGE FROM QU TO FREE
T←RESP+1
STOP TIMER
MODIFY ERROR COUNTERS, EXAMING THEM AND TAKE SUITABLE ACTION
EXIT

FIG. 4.6 (c)

ACKR

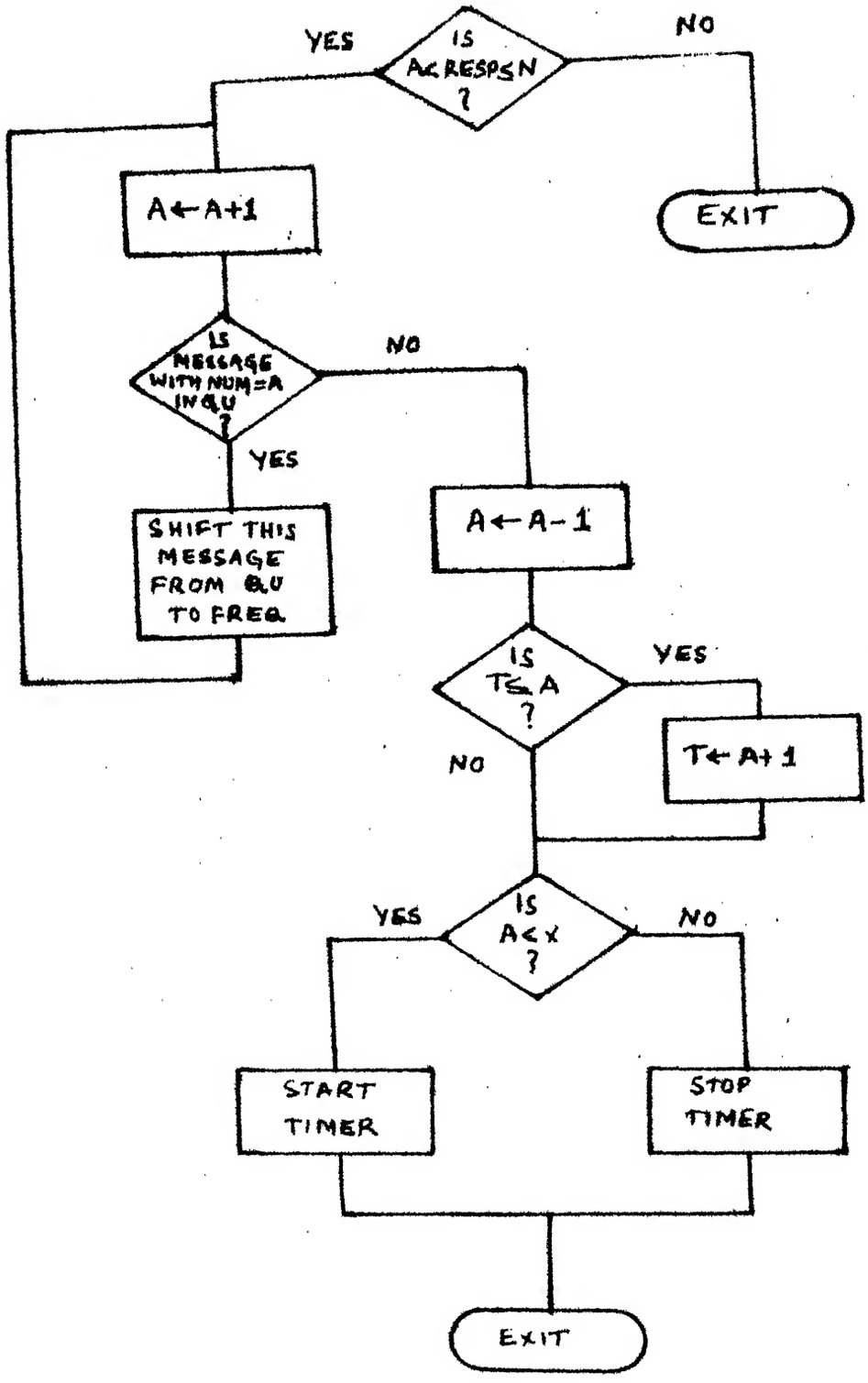

IS AC RESPSN ?
YES
NO
EXIT
A←A+1
IS MESSAGE WITH NUM=A IN Q.U ?
NO
YES
SHIFT THIS MESSAGE FROM Q.U TO FREQ.
A←A-1
IS T≤A ?
YES
NO
T←A+1
IS A<X ?
YES
NO
START TIMER
STOP TIMER
EXIT

FIG. 4.6(b)

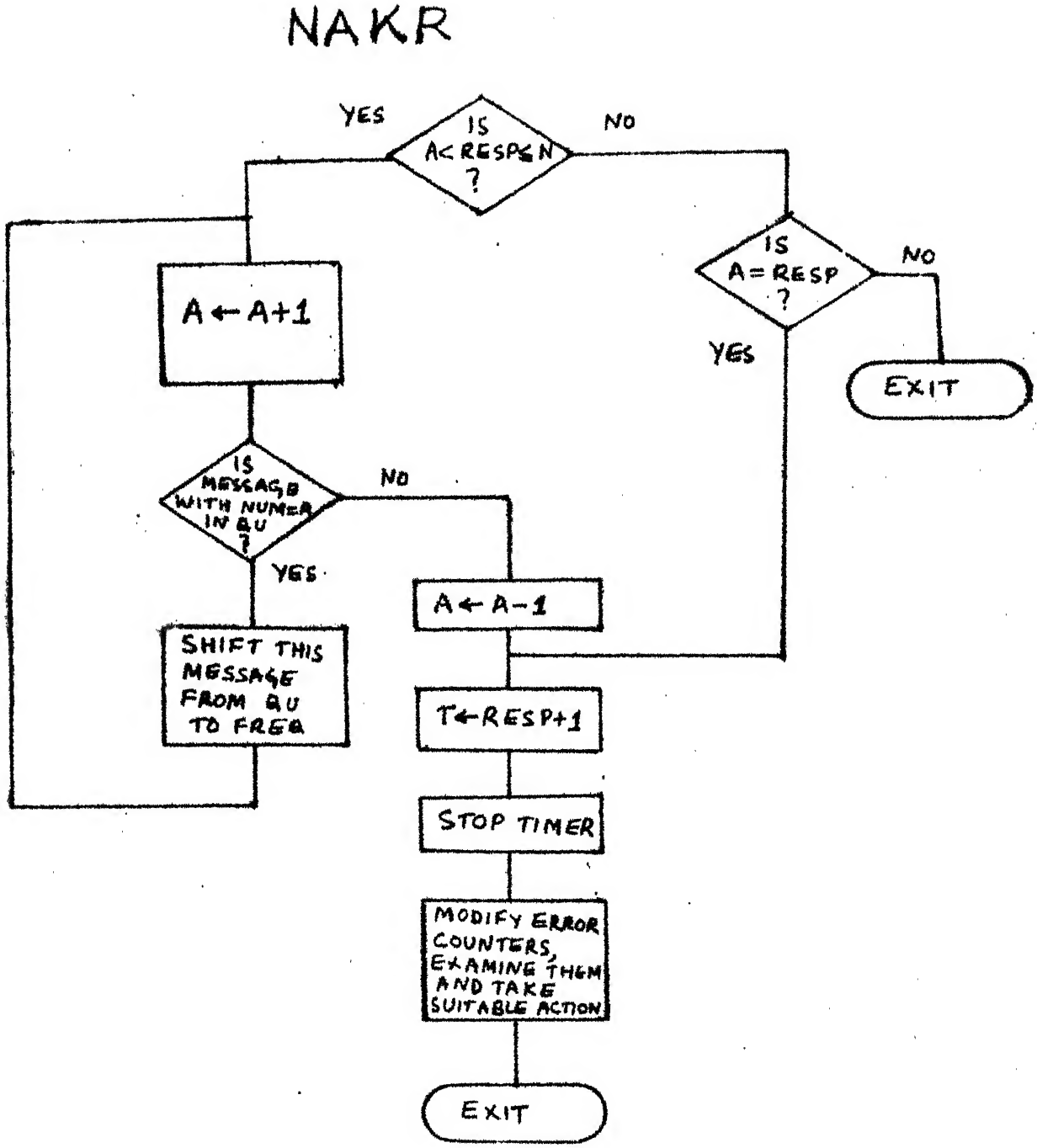

NAKR
YES
IS A<RESP&N ?
NO
IS A=RESP ?
NO
YES
EXIT
A←A+1
IS MESSAGE WITH NUMBER IN QU ?
NO
YES
A←A-1
SHIFT THIS MESSAGE FROM QU TO FREE
T←RESP+1
STOP TIMER
MODIFY ERROR COUNTERS, EXAMINE THEM AND TAKE SUITABLE ACTION
EXIT

FIG. 4.6 (c)

4-7. TIMER AND ITS INTERRUPT ROUTINE

As soon as a data message or a REP is transmitted on the link in RUNNING state, or a STACK or STRT is transmitted during the STARTING state, a timer is started and after a selected time interval an interrupt is generated if no ACK or NAK is received during this interval. The timer interval can be selected by assigning a value to a variable TIMVAL. This value is calculated by the formula

$$\text{TIMVAL} = 2\text{'s complement of } \left(\frac{\text{Interval Required}}{\text{Time base used}}\right)$$

The timer interrupt routine examines the STATE of DDCMP and takes appropriate actions as shown in Figure 4-7.

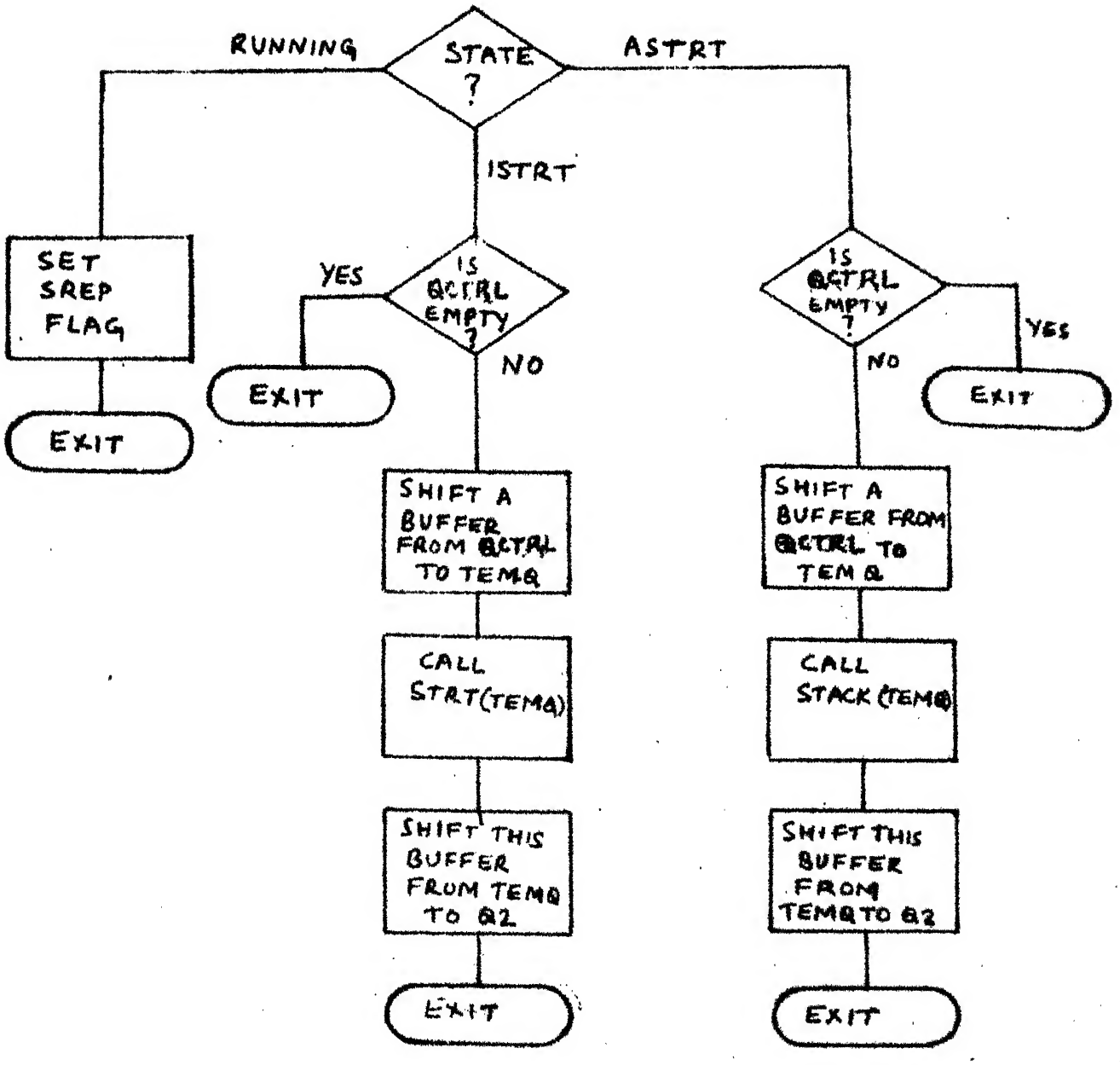


FIG 4.7 : TIMER ROUTINE

CHAPTER 5

CONCLUSION

Our aim during the current phase was to implement the lowest layer protocol on IBM-1800 and MICRO-78 and design the hardware interfaces for the MICRO to DEC link. Since DDCMP was the line protocol used by DEC and its implementation was available on the system, this protocol was chosen for our network and we implemented it on MICRO-78 and IBM-1900. The hardware interfaces designed were found flexible enough to support DDCMP without much problems. However, some other problems were discovered in the design of these interfaces and suitable changes have been made to solve these problems. Moreover, the hardware interfaces of MICRO-78 and TDC-316 though tested gave frequent problems because their fabrication was not very neat and are therefore being refabricated by a professional. The interfaces of IBM-1800 are, however, in a working condition and DDCMP has been tested on them in a loop back mode. On MICRO-78 also, DDCMP has been implemented and tested except for the interrupt routines.

The design of the MICRO to DEC interfaces had to conform to the DQ11 specifications and so it is somewhat different from the other interfaces in the network.

As soon as the hardware interfaces of MICRO-78 and TDC-316 are refabricated DDCMP can be tested on the IBM-1800-MICRO-78-DEC-10 link and subsequently, higher level protocols can be implemented to provide network facilities between IBM 1800 and

and DEC-10. For TDC-316, the software is already available, and we do not foresee any problems in connecting it to the network.

Since the hardware interfaces on IBM-1800, TDC-316 and MICRO-78 (except the MICRO to DEC side interfaces) were designed before the digital network specifications were available, their design included some features which should now be changed. One change that should be made is to remove the parity generation and checking circuits from these interfaces, since the DQ-11 interface of DEC-10 does not include this feature. Moreover, if any errors do creep in, they are taken care of by the CRC block checks specified in DDCMP. Presently, the CRC generation and checks are performed by software routines, which cause a lot of overhead. To remove this overhead, it is desirable that hardware chips are used to do this work. Another inconsistency between the DDCMP specifications and the earlier hardware interfaces is that the value of SYN character used on the MICRO-78 links to TDC-316 and IBM-1800 is 026_8 as against the value 226_8 specified by DDCMP. This inconsistency can easily be removed by making minor hardware changes.

To increase the data transmission rates, it is suggested that DMA interfaces be used for data transmission and reception. This is particularly essential on the MICRO-78. Another suggestion is that provision be made for varying the baud rate on the links.

APPENDIX I

ERROR MESSAGES

While performing queue operations, two types of fatal errors may be detected and when these errors occur, the system comes to a halt after printing some error messages. DDCMP must be reloaded when these errors occur. The error messages are:

(1) 'ATTEMPT TO SHIFT FROM AN EMPTY Q. FATAL ERROR RELOAD SYSTEM.'

This message means that at some stage an attempt was made to take a buffer from some queue which was already empty.

(2) 'SRCHQ FAILED. FATAL ERROR RELOAD SYSTEM'.

This message means that a buffer was not found in a list when a search was being made for it.

APPENDIX II

OPTIONS USED IN DQ11

The DQ11 is a high speed, double buffered communications device designed to interface the PDP-11 processor to a serial synchronous communications channel. Transmit and receive data transfers between the PDP-11 Unibus and the DQ11 are handled as non processor requests (NPRs). As an NPR device, the DQ11 provides extremely fast access to the PDP-11 unibus.

For a detailed description of DQ11, Ref. 3 may be seen. We shall give here the options on DQ11 chosen by us.

(1) Transmission speed - 10K bauds
(2) Full duplex operation
(3) No parity
(4) Two SYN characters (226_8, 226_8)
(5) Character size of 8 bits
(6) The three switch selectable control characters for program interrupts are 201_8, 005_8 and 220_8.
(7) CRC check using the polynomial $x^{16} + x^{15} + x^2 + 1$.
(8) No modem used
(9) For reception and transmission external clock is used. This clock comes from the MICRO-78 interface on a separate line.
(10) Interfacing is done in EIA mode.

REFERENCES

1. Das, D., "IITK Computer Network - MICRO-78 Hardware Interfaces"

2. DECNET : Digital Data Communications Massage Protocol (DDCMP) Specification Version 4.0

3. DQ11 Maintenance Manual

4. IBM-1800 Functional Characteristics

5. Krishna, C.V., "IITK Computer Network - IBM-1800 Hardware Interface"

6. Narayanan, N.S., "IITK Computer Network - TDC-316 Hardware Interface."